Learning to Play Games
Tutorial Lectures

Professor Simon M. Lucas
Game Intelligence Group
University of Essex, UK

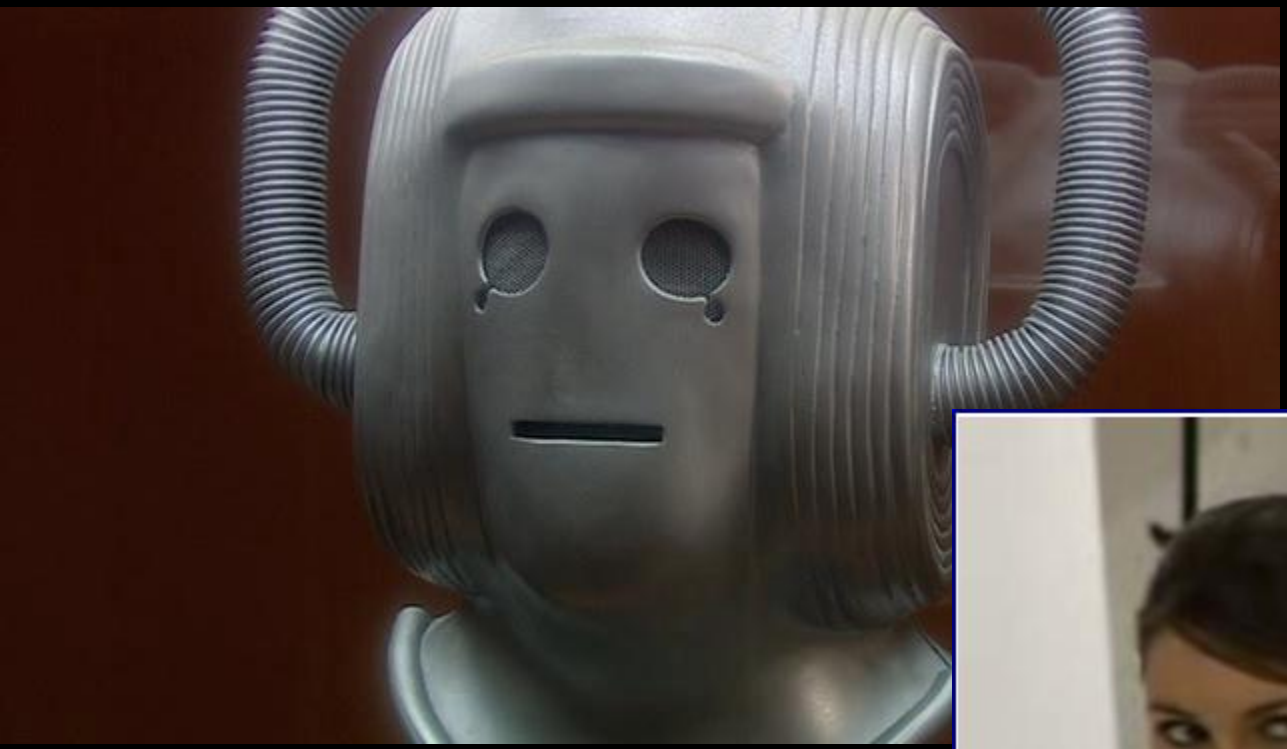
Aims

- Provide a practical guide to the main machine learning methods used to learn game strategy autonomously
- Provide insights into when each method is likely to work best
- Demonstrate Temporal Difference Learning (TDL) and Evolution in action
- Familiarity with these will help:
  - Neural networks: MLPs and Back-Propagation
  - Basics of evolutionary algorithms (evaluation, selection, reproduction/variation)

Overview

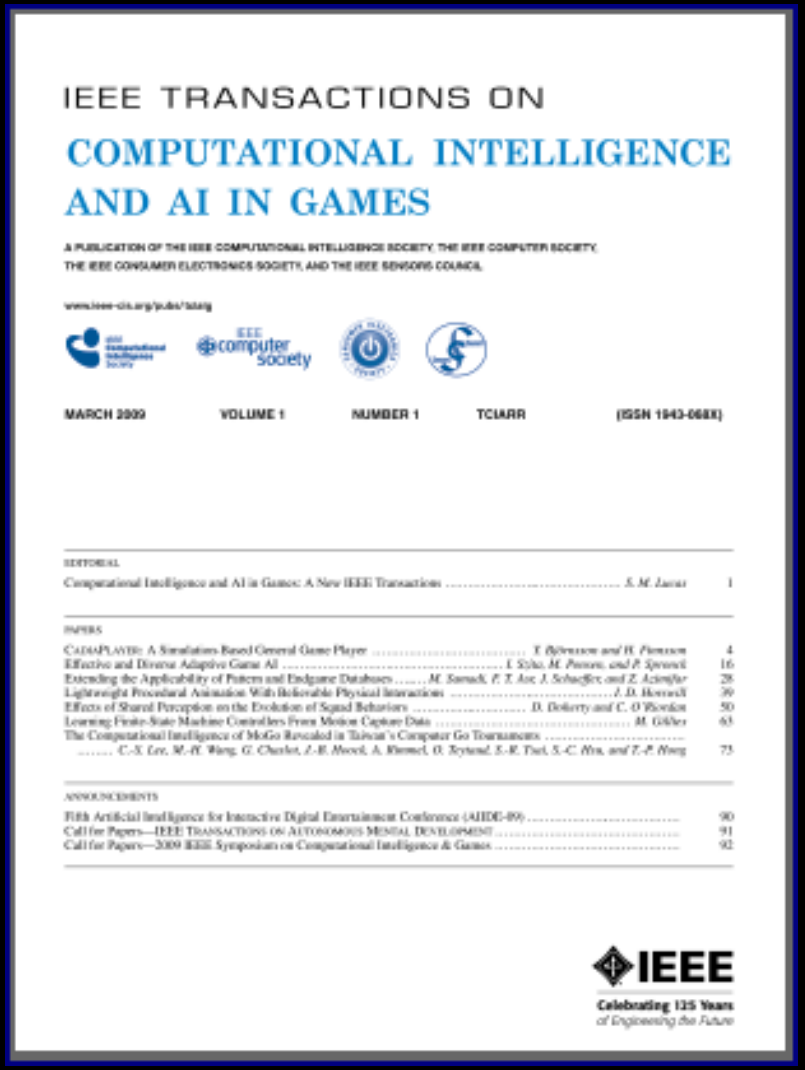
- Architecture (action selector v. value function)
- Learning algorithm (Evolution v. Temporal Difference Learning)
- Information rates
- Function approximation method
  - E.g. MLP or Table Function
  - Interpolated tables
- Sample games (Grid-world, Mountain Car, Othello, Ms. Pac-Man)

Plug and Play NPCs and Game Mashups



IEEE Transactions on Computational Intelligence and AI in Games

- Spotlights
  - Galactic Arms Race
  - Evolutionary Game Design
  - General Game Playing / Monte Carlo Tree Search
  - Bot Turing Test
  - TORCS Car Racing



IEEE TRANSACTIONS ON

COMPUTATIONAL INTELLIGENCE AND AI IN GAMES

A PUBLICATION OF THE IEEE COMPUTATIONAL INTELLIGENCE SOCIETY, THE IEEE COMPUTER SOCIETY, THE IEEE CONSUMER ELECTRONICS SOCIETY, AND THE IEEE SENSORS COUNCIL

www.ieee-cis.org/pubs/tciaig

MARCH 2009 VOLUME 1 NUMBER 1 TCIARR (ISSN 1943-068X)

EDITORIAL

Computational Intelligence and AI in Games: A New IEEE Transactions ..... S. M. Lucas 1

PAPERS

CADIAPLAYER: A Simulation-Based General Game Player ..... Y. Björnsson and H. Finnsson 4
Effective and Diverse Adaptive Game AI ..... I. Szita, M. Ponsen, and P. Spronck 16
Extending the Applicability of Pattern and Endgame Databases ..... M. Samadi, P. T. Lee, J. Schaeffer, and Z. Azimifar 28
Lightweight Procedural Animation With Believable Physical Interactions ..... J. D. Hermes 39
Effects of Shared Perception on the Evolution of Signal Behaviors ..... D. Doherty and C. O'Riordan 50
Learning Finite-State Machine Controllers From Motion Capture Data ..... M. Gökhan 63
The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments ..... C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong 73

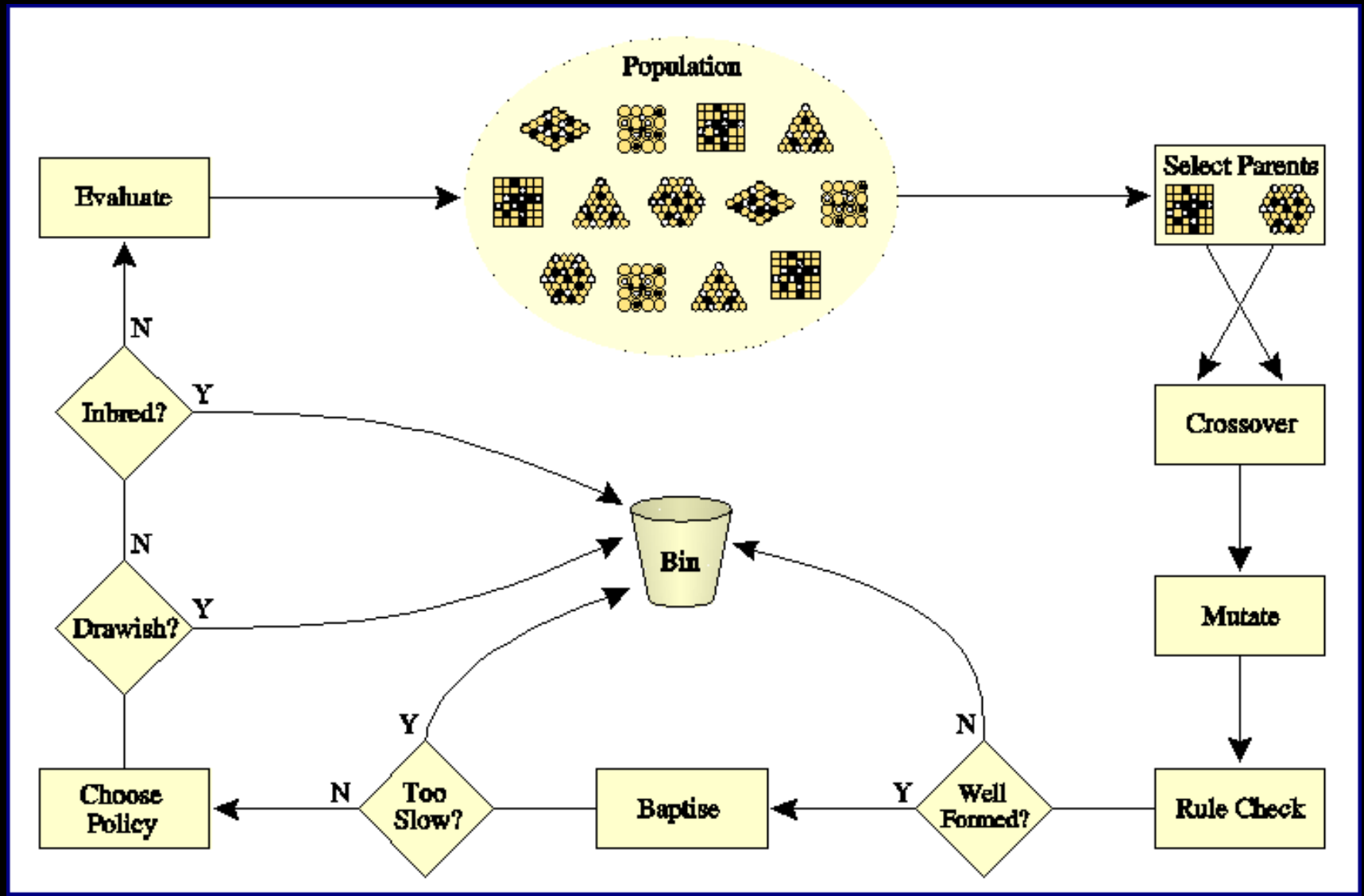
ANNOUNCEMENTS

Fifth Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-09) 90
Call for Papers—IEEE TRANSACTIONS ON AUTONOMOUS MENTAL DEVELOPMENT 91
Call for Papers—2009 IEEE Symposium on Computational Intelligence & Games 92

IEEE
Celebrating 125 Years
of Engineering the Future

Galactic Arms Race (v1, n4)
(Hastings, Guha, Stanley, UCF)

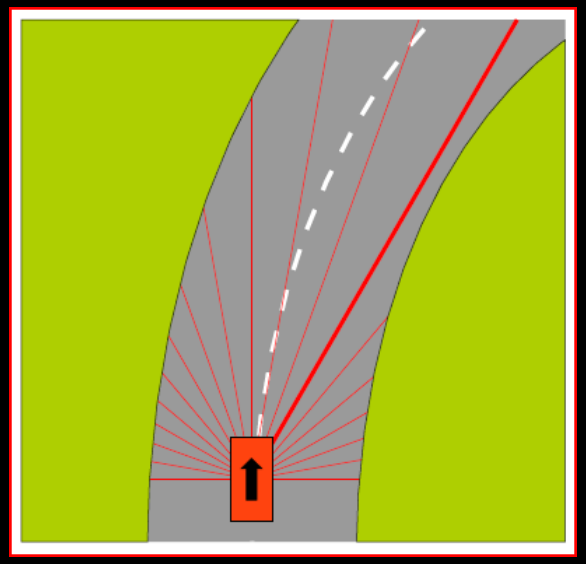
Evolving Board Games (Brown and Maire, v2 n 1)



Unreal Tournament: Pogamut Interface

TORCS Car Racing (Loiacono et al v2 n2)

(images from COBOSTAR paper, Butz and Loenneker, IEEE CIG 2009)



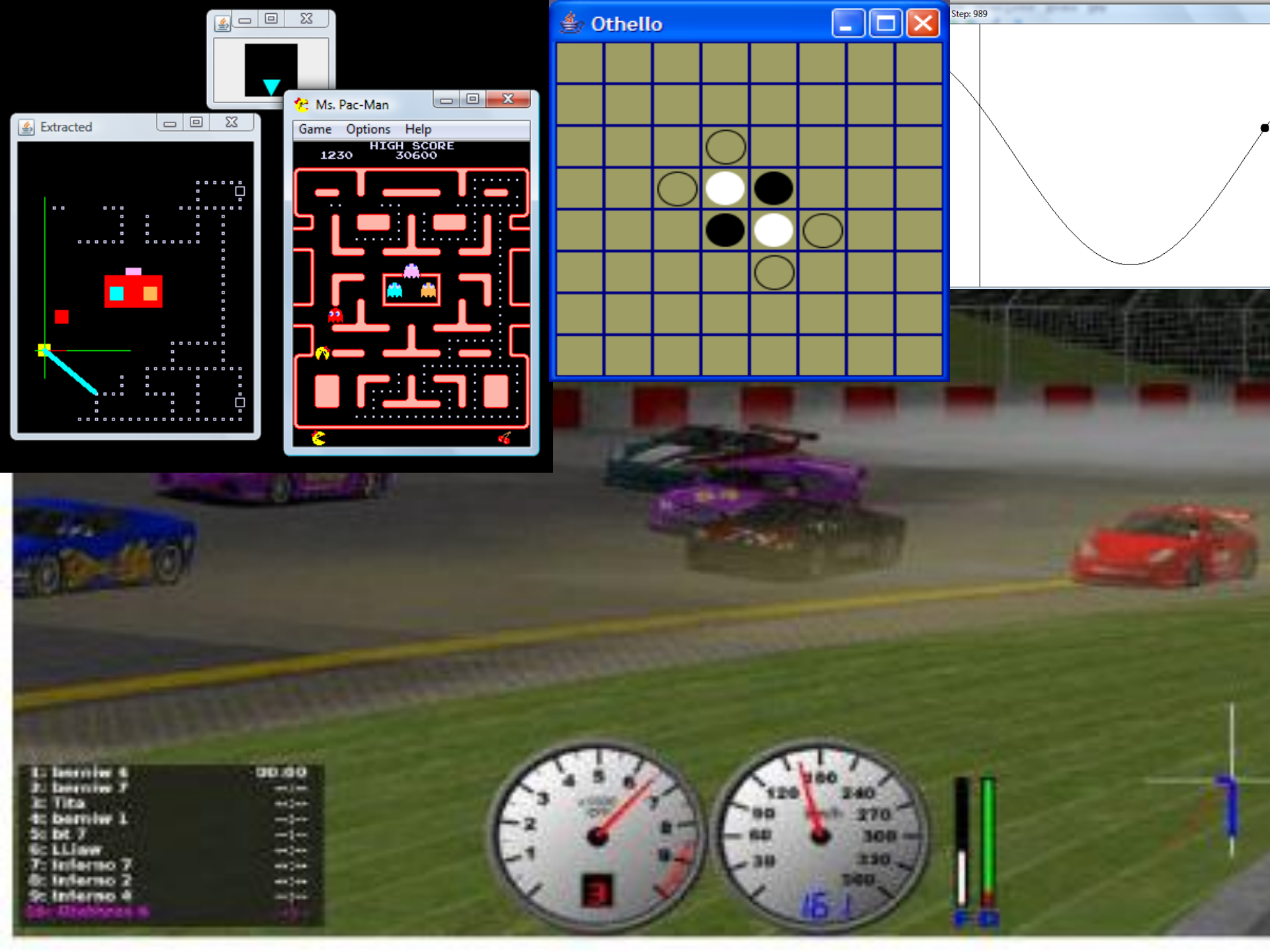
?

Action
(steer, accel.)

Video Game Competitions
Story so far
Winners tend to be hand-programmed
with some evolutionary tuning

Evaluation

- Open competitions are really important
- Example: many many papers on Pac-Man
- But in most cases the results are not directly comparable
  - Different simulator used
- How much learning has taken place?
  - How much help is the agent given (e.g. 1-ply search versus 10-ply minimax)
  - Look-ahead changes the problem complexity


Extracted
Ms. Pac-Man
Game Options Help
HIGH SCORE
1230
30600
Othello
Step: 989

Importance of Noise / Non-determinism

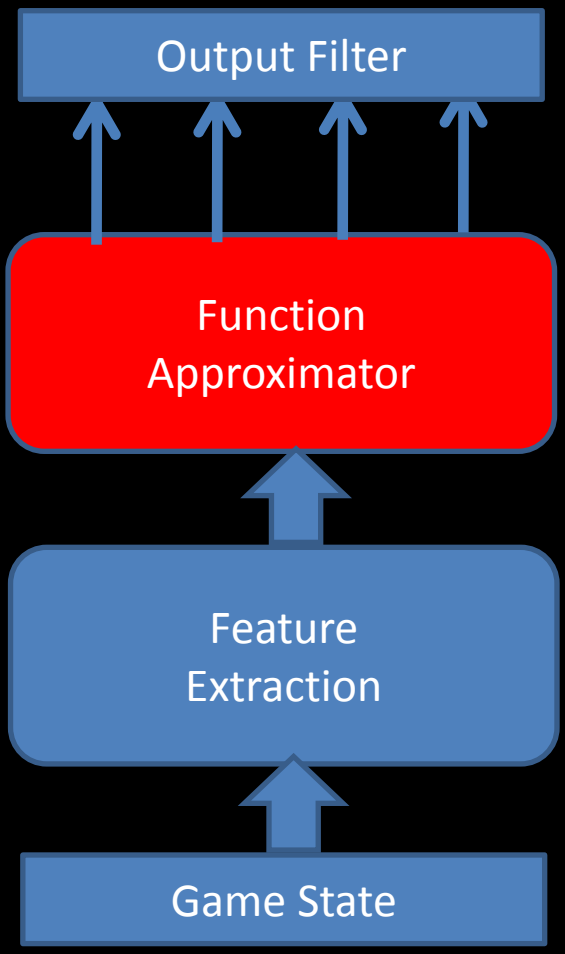
- When testing learning algorithms on games (especially single player games)
- Important that they are non-deterministic
- Otherwise evolution may evolve an implicit move sequence
- Rather than an intelligent behaviour
- Use an EA that is robust to noise
  - And always re-evaluate survivors

Architecture

- Where does the computational intelligence fit in to a game playing agent?
- Two main choices
  - Value function
    - E.g. [TD-Gammon], [Neuro-Gammon], [Blondie]
  - Action selector
    - [NERO], [MENACE]
- We'll see how this works in a simple grid world

Action Selector

- Maps observed current game state to desired action
- Multiple output F.A.
- For
  - No need for internal game model
  - Fast operation when trained
- Against
  - More training iterations needed (more parameters to set)
  - May need filtering to produce legal actions

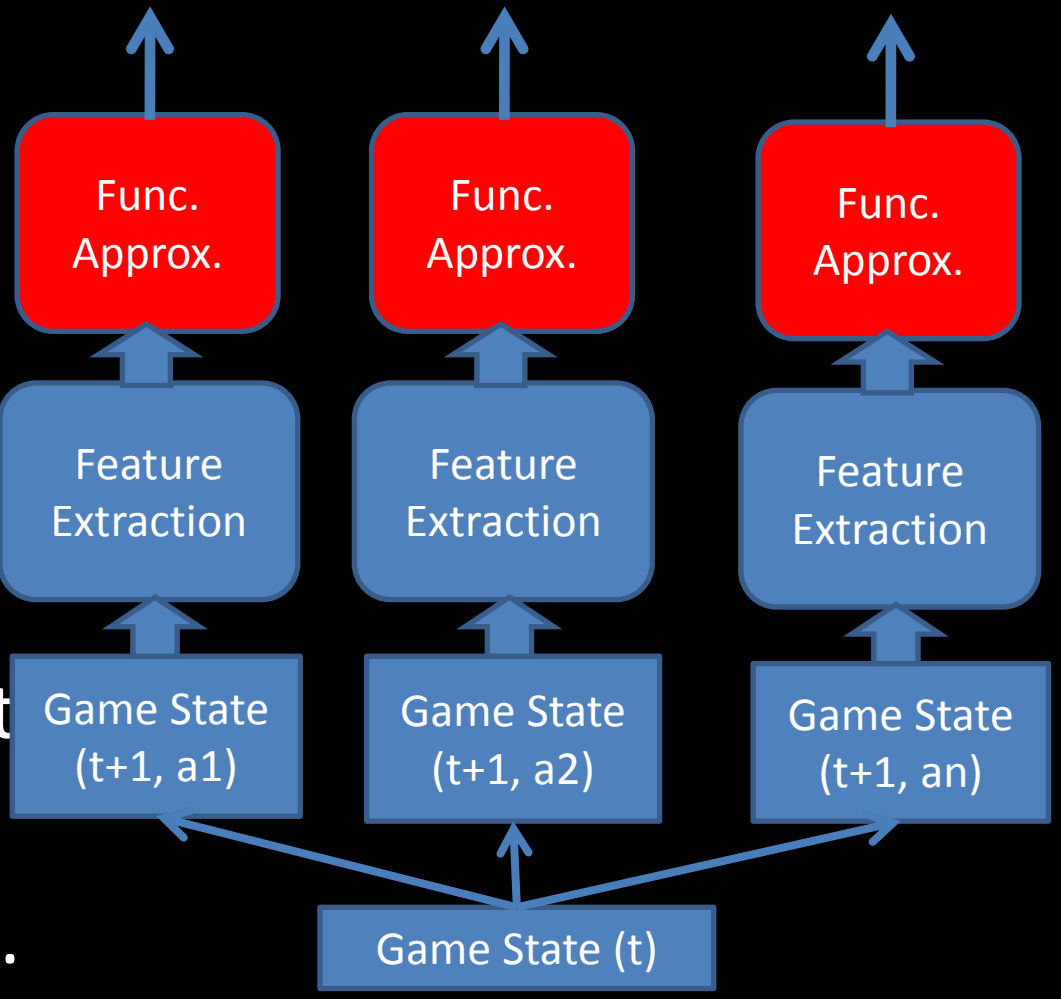


State Value Function

- Hypothetically apply possible actions to current state to generate set of possible future states
- Evaluate these using value function
- Pick the action that leads to the most favourable state
- For
  - Easy to apply, learns *relatively* quickly
- Against
  - Need a model of the system

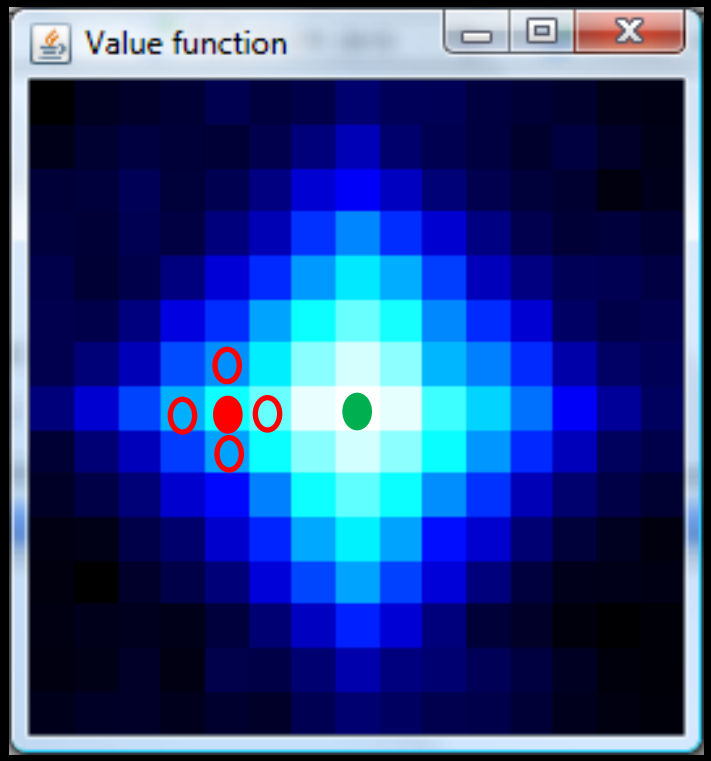
State Value Diagram

- Game state projected to possible future states
- These are then evaluated
- Choose action that leads to best value
- Single output F.A.



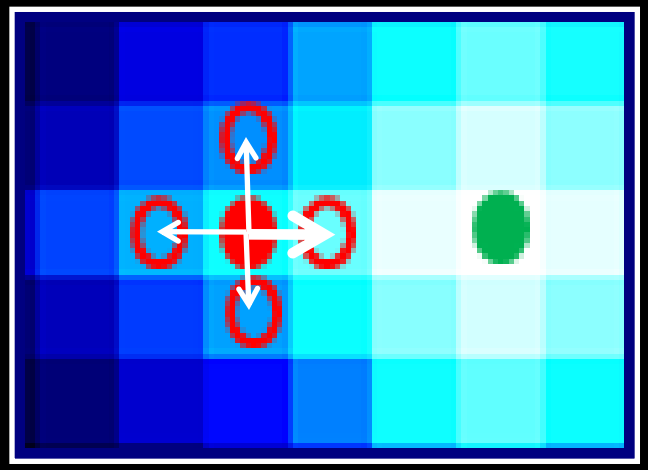
Grid World

- n x n grid (toroidal i.e. wrap-around)
- Green disc: goal state
- Red disc: current state
- Actions: up, down, left, right
- Red circles: possible next states
- Example uses 15 x 15 grid



Grid World: State Value Approach

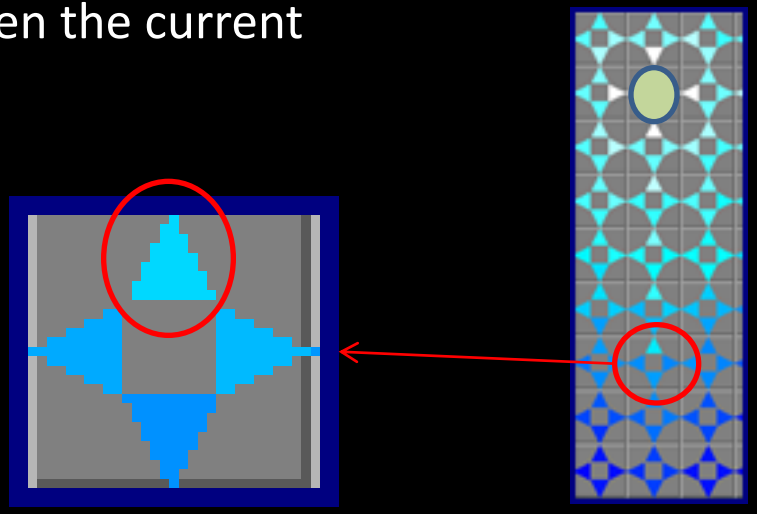
- State value: consider the four states reachable from the current state by the set of possible actions
  - choose action that leads to highest value state



Grid World: Action Selection Approach

State-Action Value

- Take the action that has the highest value given the current state



Learning Algorithms

- Next we'll study the main algorithms for learning the parameters of a game-playing system
- These are temporal difference learning (TDL) and Evolution (or Co-Evolution)
- Both can learn to play games given no expert knowledge

Temporal Difference Learning (TDL)

- Can learn by self-play
- Essential to have some reward structure
  - This may follow directly from the game
- Learns during game-play
- Uses information readily available (i.e. current observable game-state)
- Often learns faster than evolution, but may be less robust.
- Function approximator must be trainable

Sample TDL Algorithm: TD(0) (adapted from [RL])

typical alpha: 0.1

π: policy; choose rand move 10% of time
else choose action leading to best state

Algorithm 1: On-line TD(0) adapted from Sutton and Barto

INITIALIZE $V(s)$ arbitrarily, for all $s \in S$
for *each episode* **do**
    Initialize s to start state
    (could be random start state)
    for *each step in episode* **do**
        $a \leftarrow$ action given by π for s
        Take action a, observe reward r, and next state s'
        $\delta \leftarrow r + V(s') - V(s)$
        $V(s) \leftarrow V(s) + \alpha\delta$
    end
end

(Co) Evolution

- Evolution / Co-evolution (vanilla form)
- Use information from game results as the basis of a fitness function
- These results can be against some existing computer players
  - This is standard evolution
- Or against a population of simultaneously evolving players
  - This is called co-evolution
- Easy to apply
- But wasteful: discards *so much* information

(1+lambda) Evolution Strategy (ES)

```
INITIALIZE (Pop, μ + λ individuals)
while running do
    for i=1 to (μ + λ) do
        EVALUATE (Pop[i])
    end
    Sort on best fitness first, breaking ties randomly
    Parents are Pop[1] ... Pop[μ]
    for i=μ + 1 to (μ + λ) do
        Pop[i] ← MUTATED COPY (RANDOMLY SELECTED PARENT)
    end
end
```

Co-evolution (single population)

Evolutionary algorithm: rank them using a league

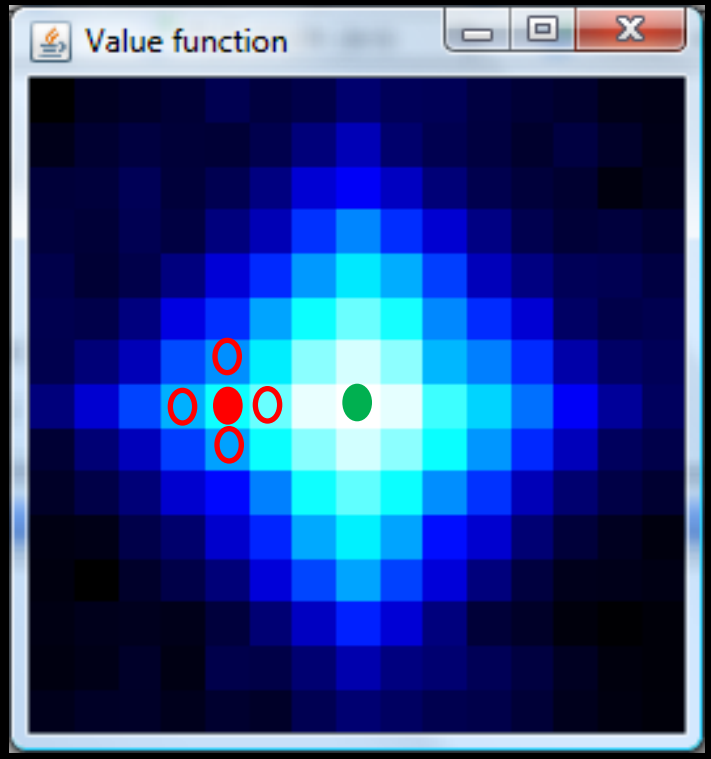
LEAGUE TABLE — BARCLAYS PREMIER LEAGUE

Updated 17:53 17th May 2009

			Home					Away						
POS		NAME	P	W	D	L	F	A	W	D	L	F	A	GD	PTS
1	-	Man Utd	37	16	2	1	43	13	11	4	3	24	11	+43	87
2	-	Liverpool	37	11	7	0	38	12	13	4	2	36	14	+48	83
3	-	Chelsea	37	11	6	2	33	12	13	2	3	32	10	+43	80
4	-	Arsenal	37	10	5	3	27	15	9	7	3	37	21	+28	69
5	-	Everton	37	8	6	5	31	20	8	6	4	22	17	+16	60
6	-	Aston Villa	37	6	9	3	26	21	10	2	7	27	27	+5	59
7	-	Fulham	37	11	3	4	28	14	3	8	8	11	18	+7	53
8	-	Tottenham	37	10	5	4	21	10	4	4	10	23	32	+2	51
9	-	West Ham Utd	37	8	2	8	21	21	5	7	7	19	23	-4	48
10	-	Manchester City	37	12	0	6	39	18	2	5	12	18	32	+7	47
11	-	Stoke City	37	10	5	4	22	15	2	4	12	15	36	-14	45
12	-	Wigan Athletic	37	7	5	6	16	18	4	4	11	17	27	-12	42
13	-	Bolton	37	7	5	7	21	21	4	3	11	20	31	-11	41

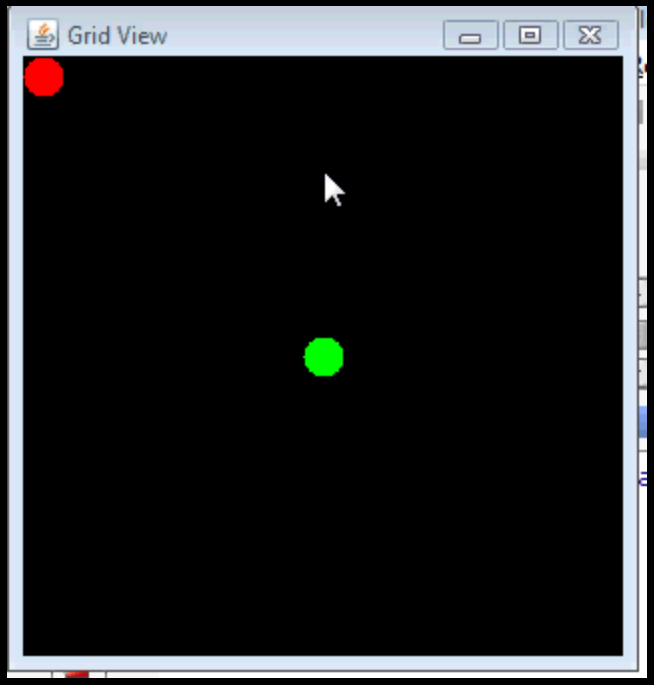
Grid World: TDL(0), State Values

- Example uses 15 x 15 grid
- Maximum number of iterations was set to 450 (twice the number of squares on the grid)
- The reward structure: -1 everywhere apart from the goal; 0 at the goal
- alpha = 0.1, epsilon = 0.1
- The diagram shows a successfully learned state value table



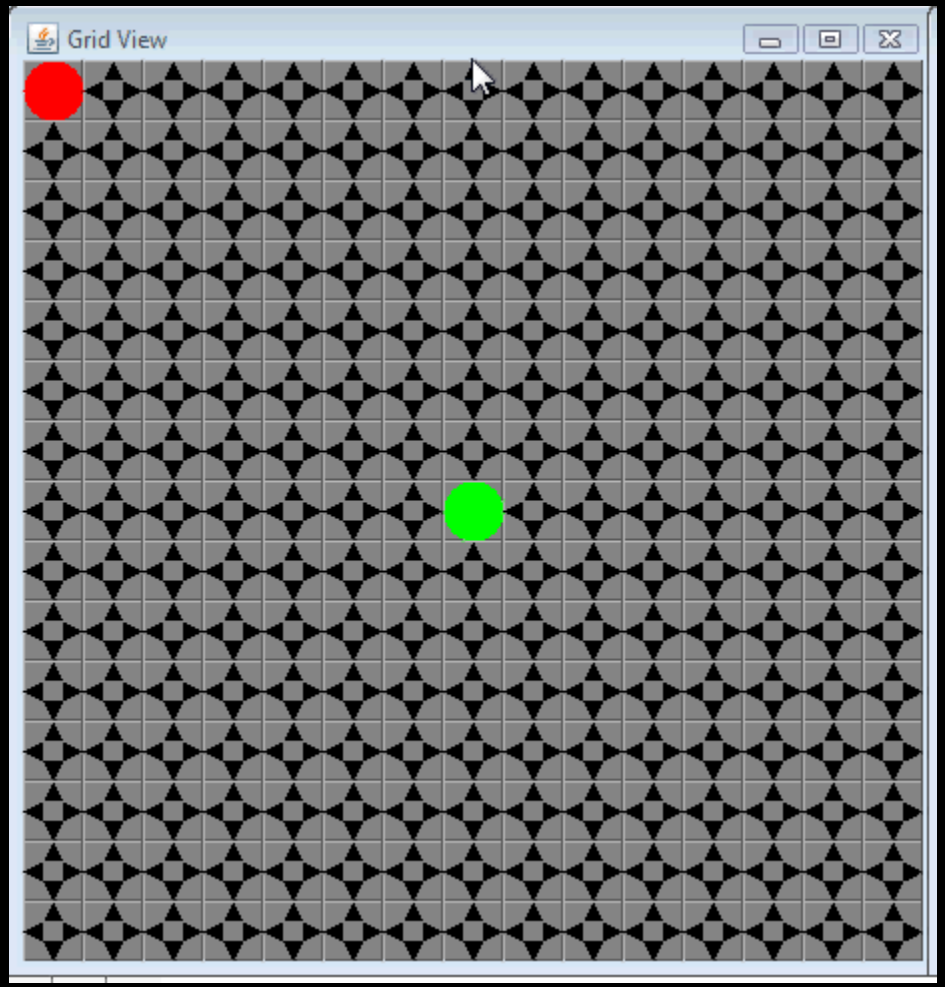
Video: TD(0) Learns a State Value Function for the Grid Problem

Play video

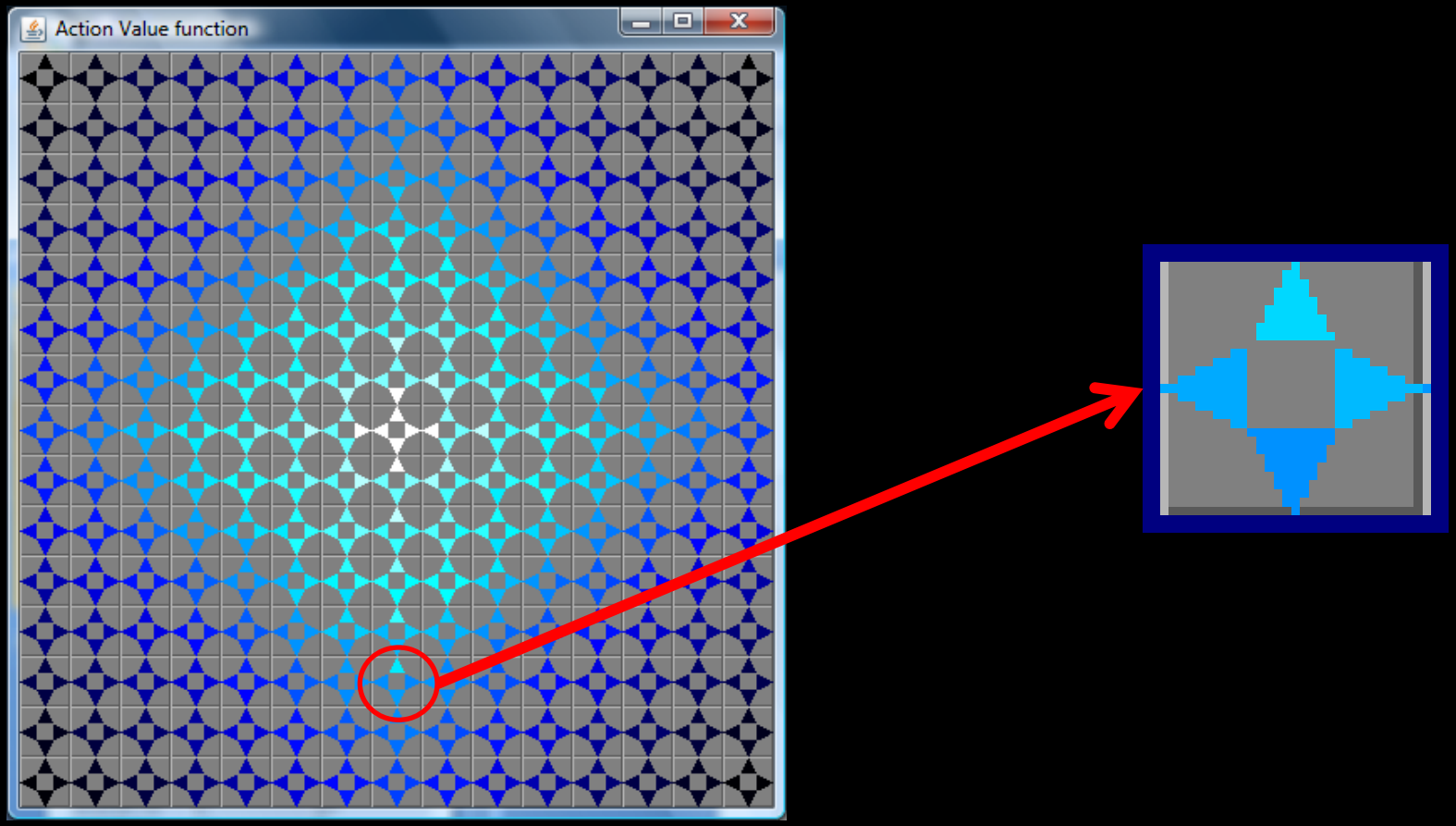


Video: TD(0) Learns a State-Action Value Table for the Grid Problem

Play Video



Learned State-Action Values (after 4,000 iterations)

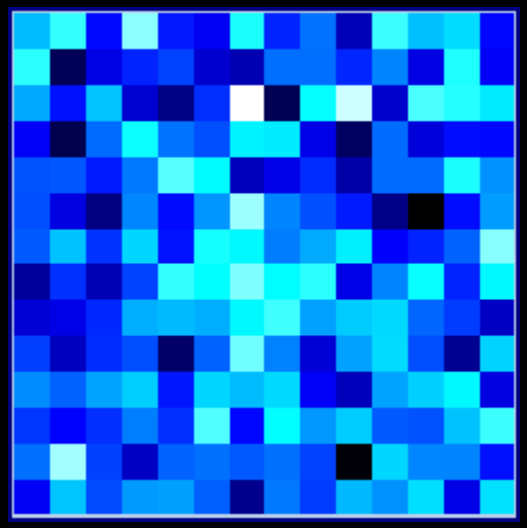


Evolving State Tables

- Interesting to make a direct comparison for the grid problem between TDL and evolution
- For the fitness function we measure some aspect of the performance
- E.g. average number of steps taken per episode given a set of start points
- Or number of times goal was found given a fixed number of steps

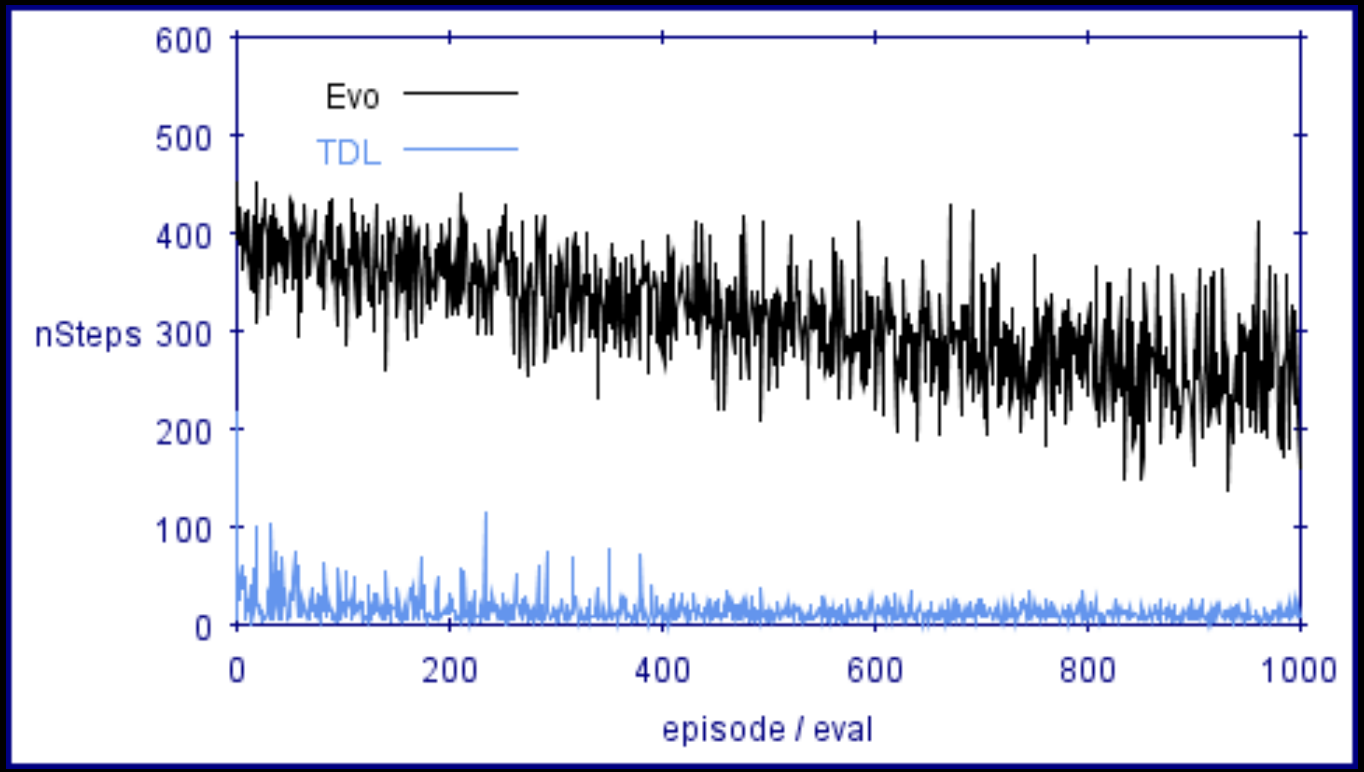
Evolving a State Table for the Grid Problem

- This experiment ran for 1000 fitness evaluations
  - fitness function: average #steps taken over 25 episodes
  - used [CMA-ES] – a powerful evolutionary strategy
  - very poor results (final fitness approx 250)
  - sample evolved table



TDL versus Evolution
Grid Problem, State Table

- TDL greatly outperforms evolution on this experiment



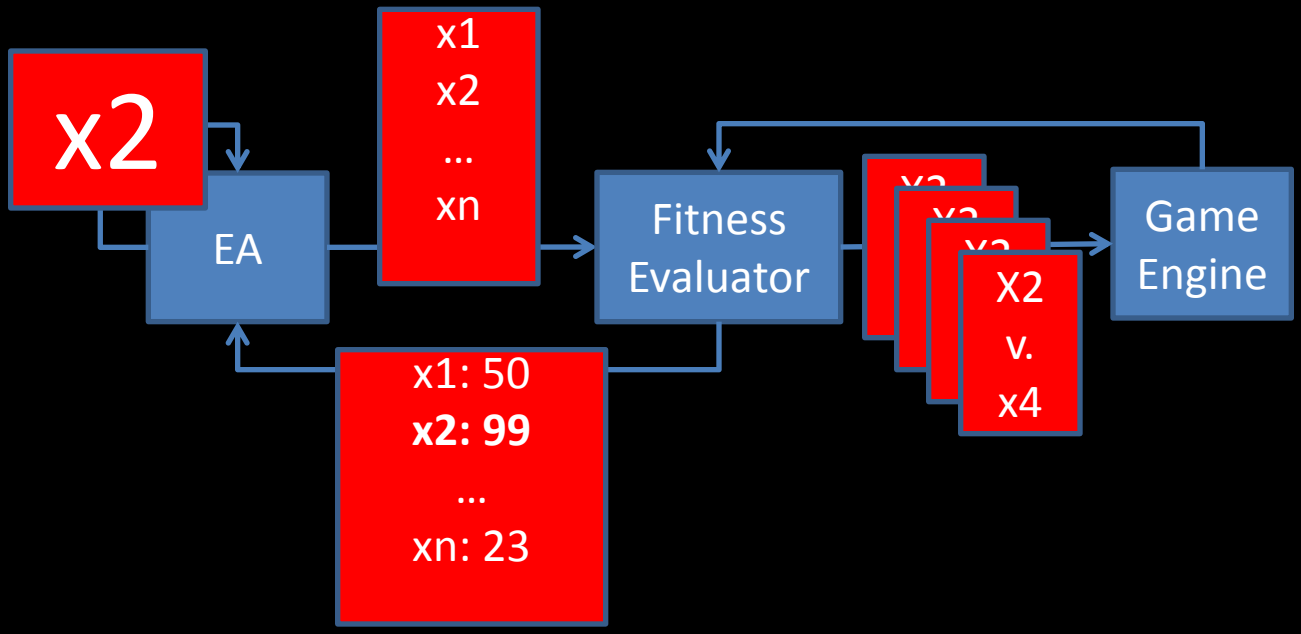
Information Rates

Information Rates

- Simulating games can be expensive
- Interesting to observe information flow
- Want to make the most of that computational effort
- Interesting to consider bounds on information gained per episode (e.g. per game)
- Consider upper bounds
  - All events considered equiprobable

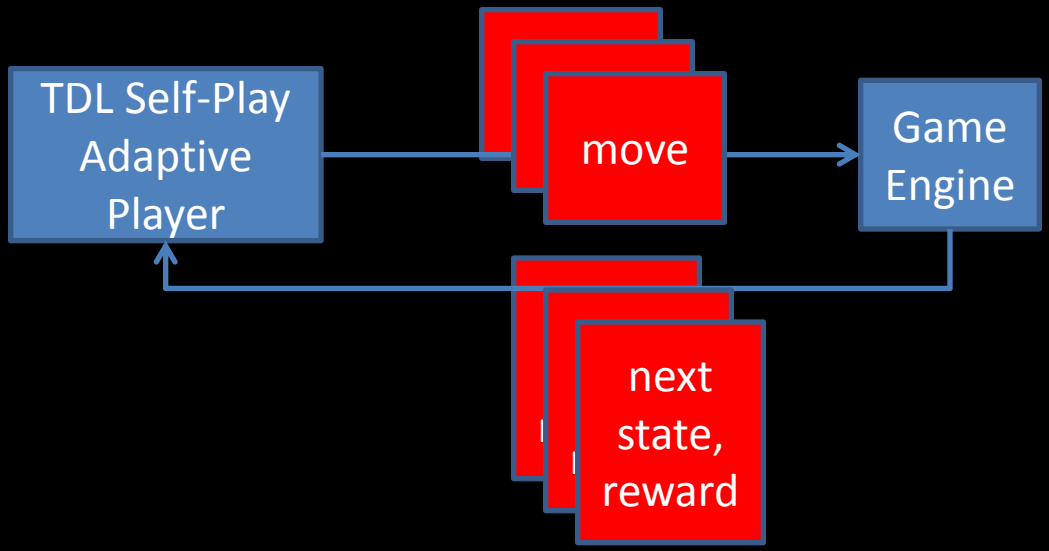
Learner / Game Interactions (EA)

- Standard EA: spot the information bottleneck



Learner / Game Interactions (TDL)

- Temporal difference learning exploits much more information
  - Information that is freely available



Evolution

- Suppose we run a co-evolution league with 30 players in a round robin league (each playing home and away)
- Need n(n-1) games
- Single parent: pick one from n
- log_2(n)
- Information rate:

$$I_c = \frac{\log_2 n}{n(n-1)}$$

n	$I_c(\mathrm{bg}^{-1})$
2	0.500
5	0.12
10	0.037
30	0.006

TDL

- Information is fed back as follows:
  - 1.6 bits at end of game (win/lose/draw)
- In Othello, 60 moves
- Average branching factor of 7
  - 2.8 bits of information per move
  - 60 * 2.8 = 168
- Therefore:
  - Up to nearly 170 bits per game (> 20,000 times more than co-evolution for this scenario)
  - (this bound is very loose – why?)

How does this relate to reality?

- Test this with a specially designed game
- Requirements for game
  - Simple rules, easy and fast to simulate
  - Known optimal policy
    - that can be expressed in a given number of bits
  - Simple way to vary game size
- Solution: treasure hunt game

Treasure Hunt Game

- Very simple:
  - Take turns to occupy squares until board is full
  - Once occupied a square is retained
  - Squares either have a value of 1 or 0 (beer or no beer)
  - This value is randomly assigned but then fixed for a set of games
  - Aim is to learn which squares have treasure and then occupy them

Game Agent

- Value function: weighted piece counter
- Assigns a weight to each square on the board
- At each turn play the free square with the highest weight
- Optimal strategy:
  - Any weight vector where every treasure square has a higher value than every non-treasure square
- Optimal strategy can be encoded with *n* bits
  - (for a board with *n* squares)

Evolution against a random player (1+9) ES

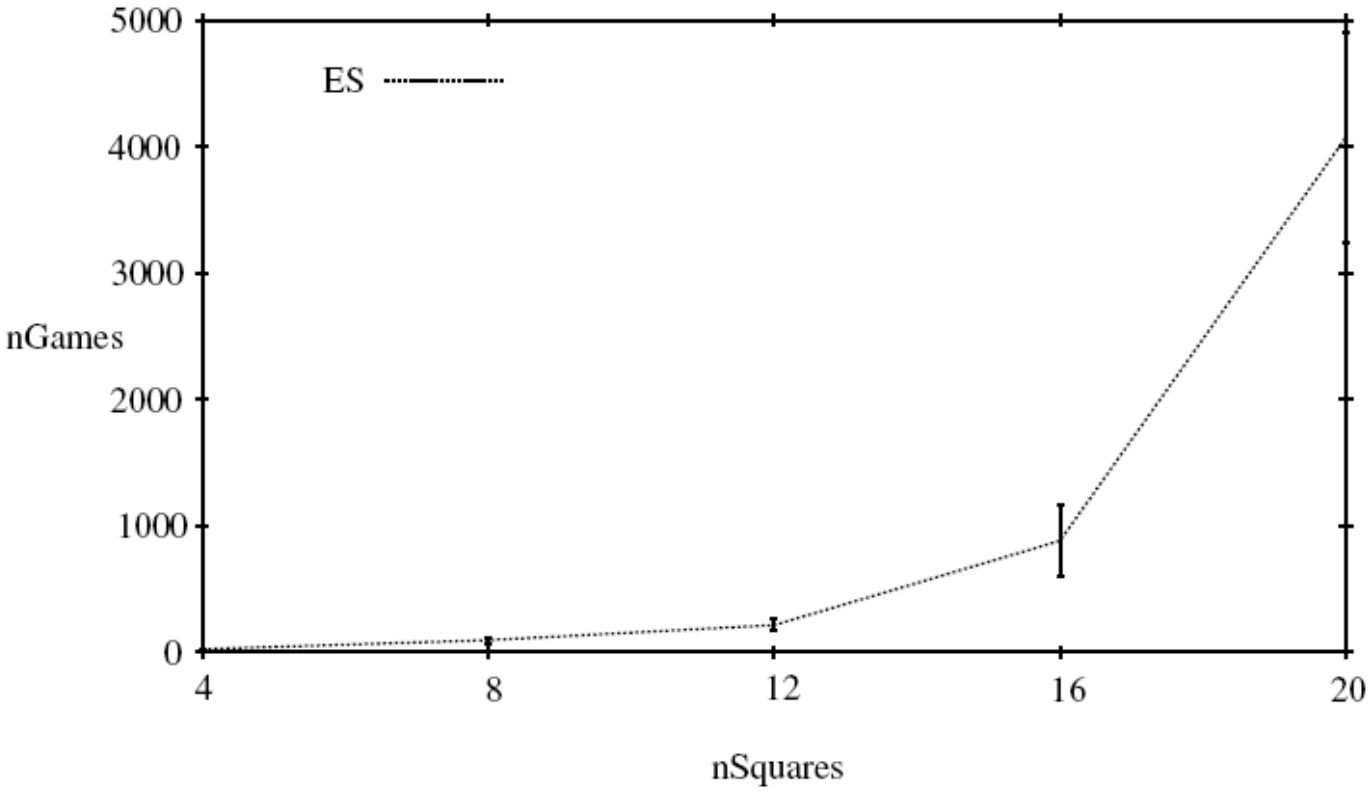


Fig. 2. Number of games against a random player required for standard evolution to learn the optimal policy, plotted against the number of squares on the board.

Co-evolution: (1 + (np-1)) ES

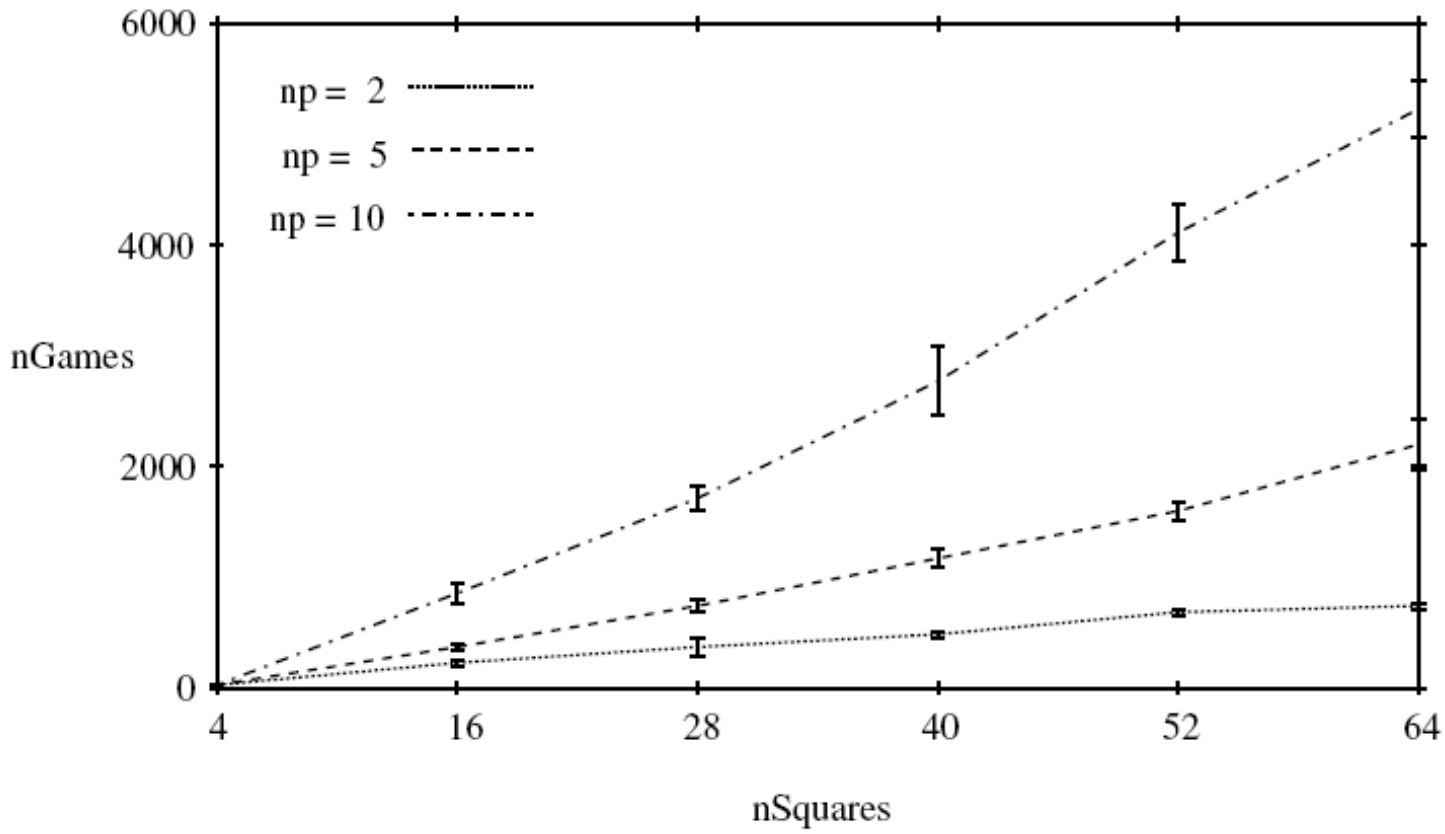


Fig. 3. Number of league games required for co-evolution to learn the optimal policy, plotted against the number of squares on the board.

TDL

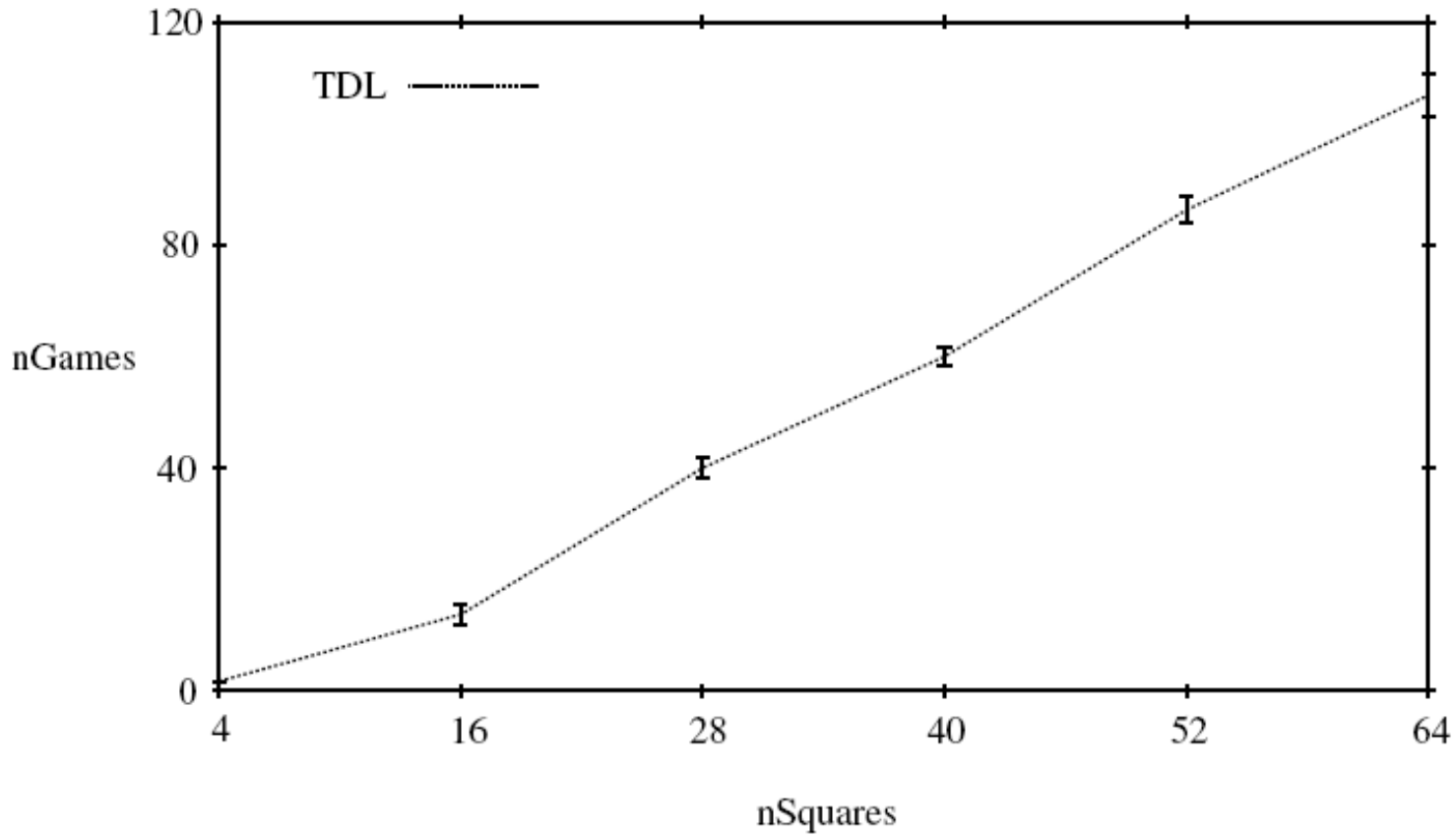


Fig. 4. Number of self-play games required to learn the optimal policy plotted against the number of squares on the board.

Results (64 squares)

method	bg^{-1}	mean (s.e.)	pred.	ratio
coev (2)	0.500	724 (55)	128	5.7
coev (5)	0.12	2188 (233)	533	4.1
coev (10)	0.037	5223 (276)	1729	3.0
TDL	166	107 (4.5)	0.4	278
TDL'	1.6	107 (4.5)	40	2.7

TABLE II

Summary of Information Rates

- A novel and informative way of analysing game learning systems
- Provides limits to what can be learned in a given number of games
- Treasure hunt is a very simple game
- WPC has independent features
- When learning more complex games actual rates will be much lower than for treasure hunt
- Further reading: [InfoRates]

Function Approximation

Function Approximation

- For small games (e.g. OXO) game state is so small that state values can be stored directly in a table
- For more complex games this is simply not possible e.g.
  - Discrete but large (Chess, Go, **Othello, Pac-Man**)
  - Continuous (Car Racing, Modern video games)
- Therefore necessary to use a function approximation technique

Function Approximators

- Multi-Layer Perceptrons (MLPs)
- N-Tuple systems
- Table-based
- All these are *differentiable, and trainable*
- Can be used either with evolution or with temporal difference learning
  - but which approximator is best suited to which algorithm on which problem?

Multi-Layer Perceptrons

- Very general
- Can cope with high-dimensional input
- Global nature can make forgetting a problem
  - Adjusting the output value for particular input point can have far-reaching effects
  - This means that MLPs can be quite bad at forgetting previously learned information
- Nonetheless, may work well in practice

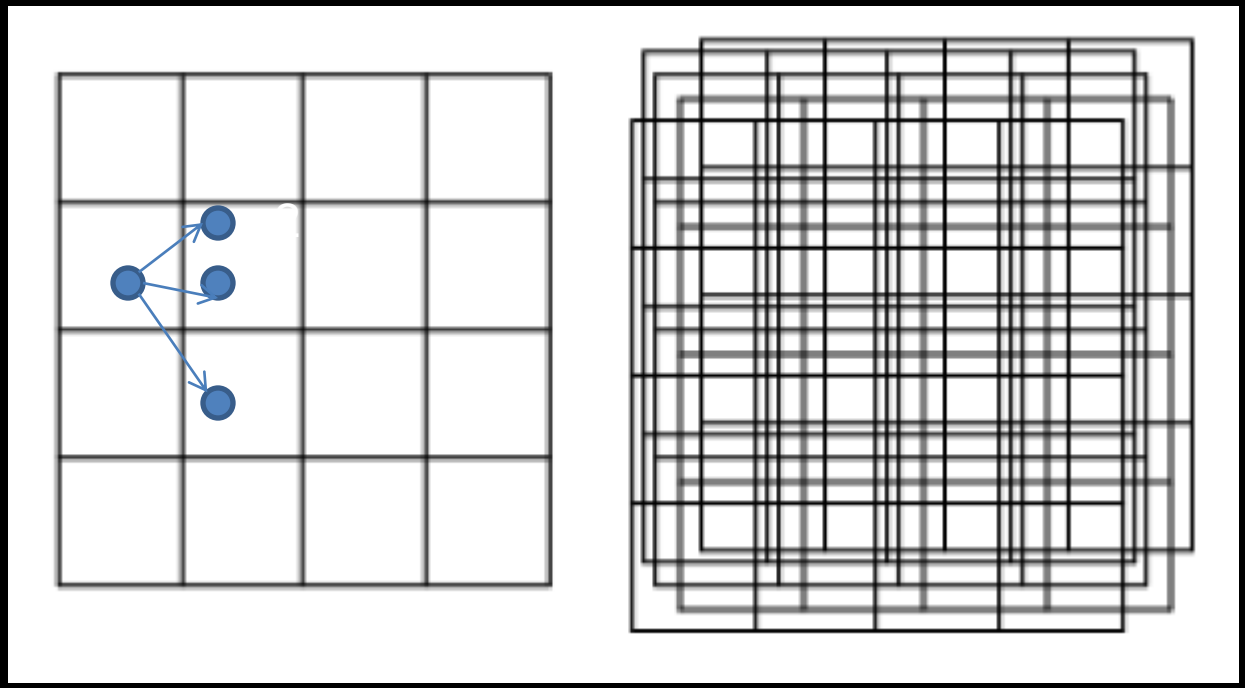
NTuple Systems

- W. Bledsoe and I. Browning. *Pattern recognition and reading by machine.* In Proceedings of the EJCC, pages 225 232, December 1959.
- Sample n-tuples of discrete input space
- Map sampled values to memory indexes
  - Training: adjust values there
  - Recognition / play: sum over the values
- Superfast
- Related to:
  - Kernel trick of SVM (non-linear map to high dimensional space; then linear model)
  - Kanerva's sparse memory model
  - Also similar to Michael Buro's look-up table for Logistello

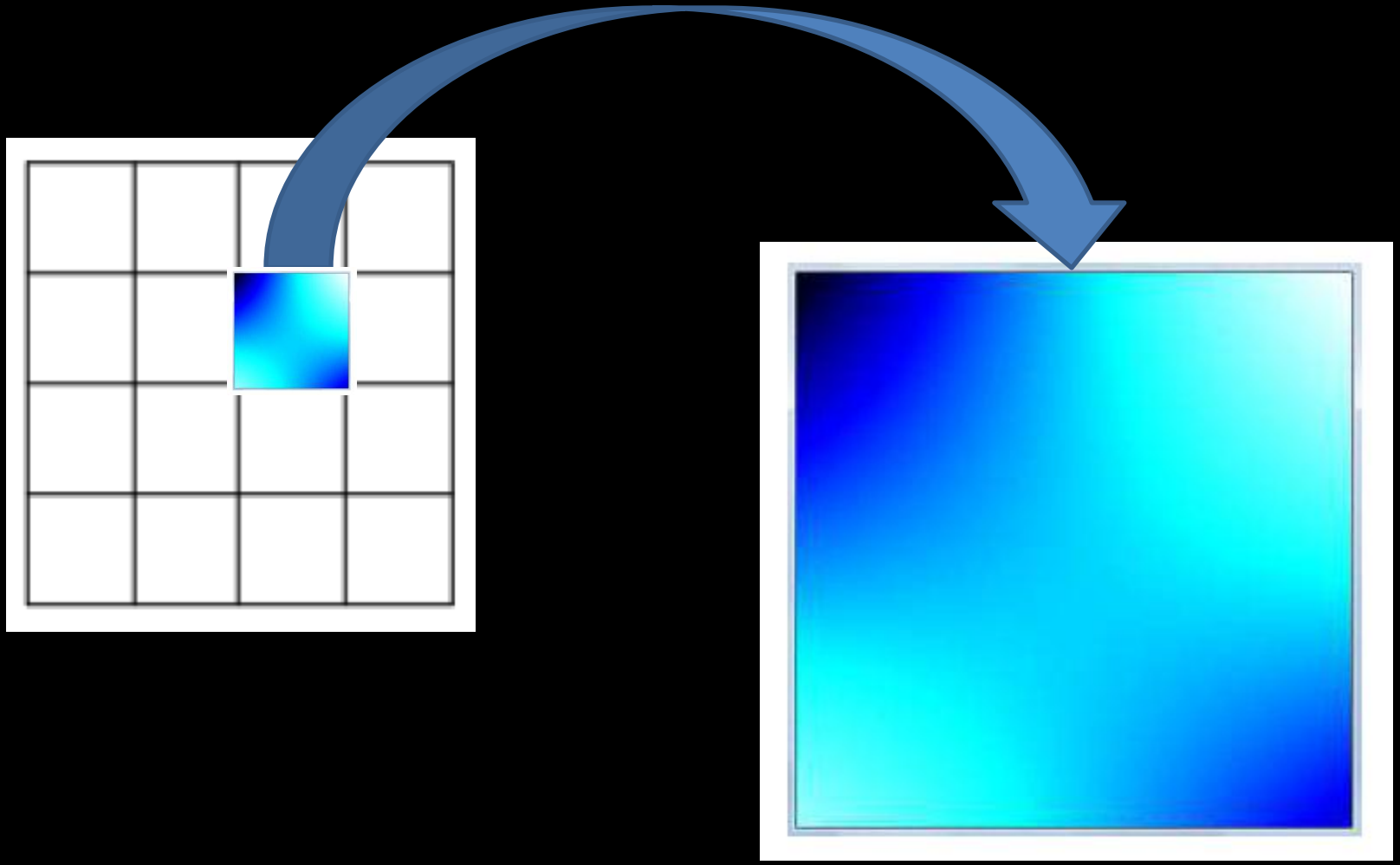
Table-Based Systems

- Can be used directly for discrete inputs in the case of small state spaces
- Continuous inputs can be discretised
- But table size grows exponentially with number of inputs
- Naïve is poor for continuous domains
  - too many flat areas with no gradient
- CMAC coding improves this (overlapping tiles)
- Even better: use interpolated tables
- Generalisation of bilinear interpolation used in image transforms

Table Functions for Continuous Inputs Standard (left) versus CMAC (right)



Interpolated Table



Bi-Linear Interpolated Table

- Continuous point p(x,y)
  - x and y are discretised, then residues r(x) r(y) are used to interpolate between values at four corner points
  - q_l (x)and q_u(x) are the upper and lower quantisations of the continuous variable x
- N-dimensional table requires 2^n lookups

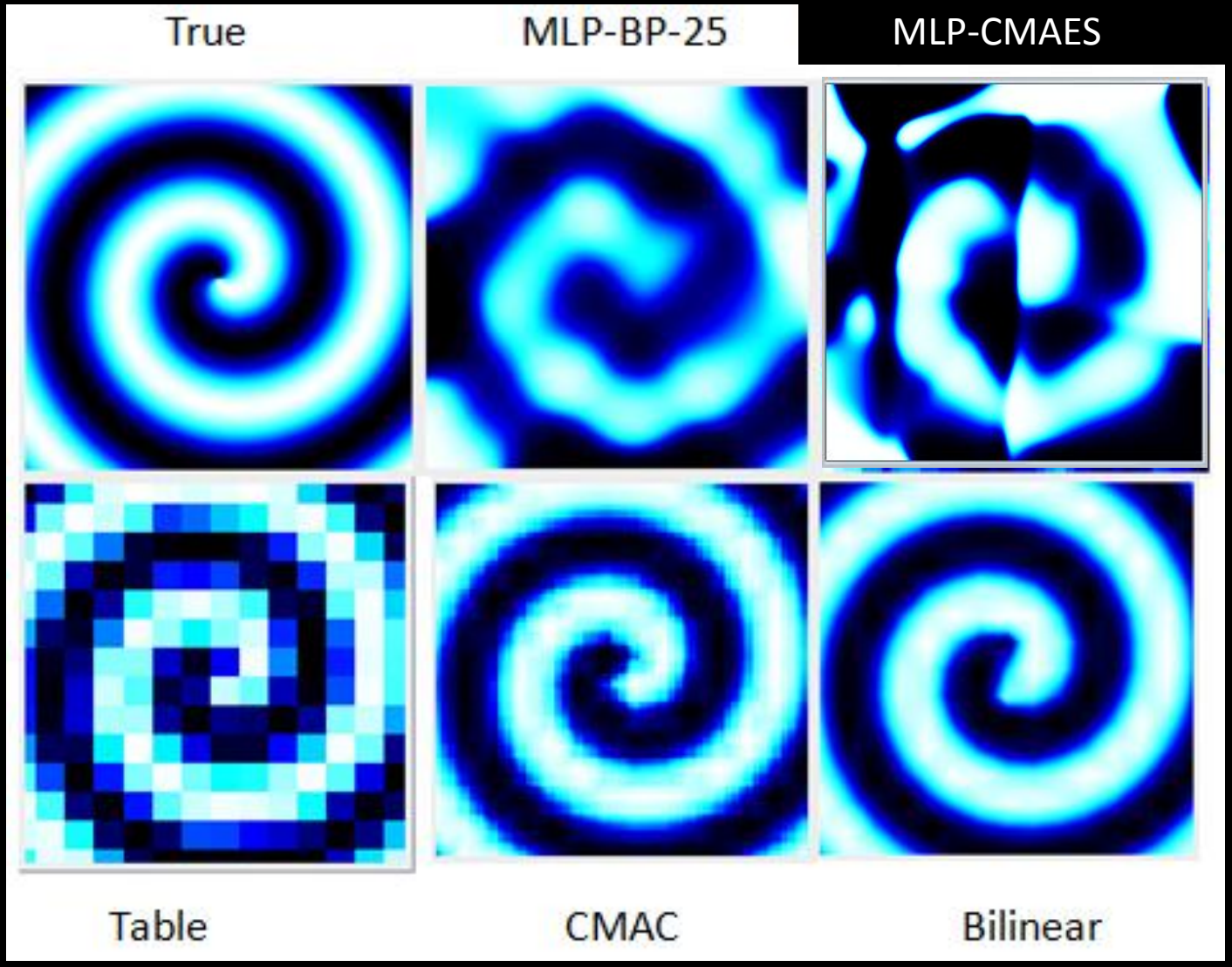
$$
\begin{aligned}
f_t(x,y) &= (1-r(x))(1-r(y))t[q_l(x)][q_l(y)] \\
&+ r(x)(1-r(y))t[q_u(x)][q_l(y)] \\
&+ (1-r(x))r(y)t[q_l(x)][q_u(y)] \\
&+ r(x)r(y)t[q_u(x)][q_u(y)]
\end{aligned}
$$

Supervised Training Test

- Following based on 50,000 one-shot training samples
- Each point randomly chosen from uniform distribution over input space
- Function to learn: continuous spiral (r and theta are the polar coordinates of x and y)

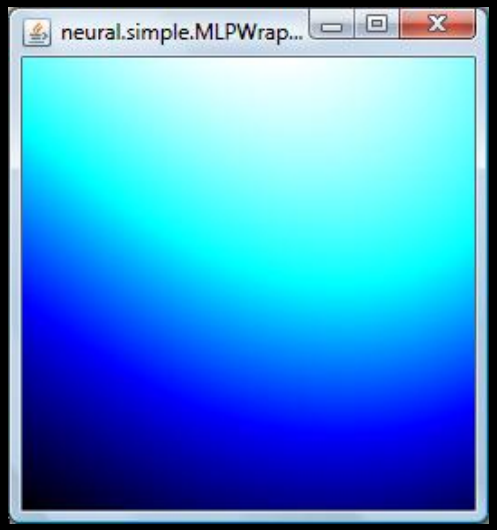
$$f(x, y) = \sin(\theta + r\pi\omega)$$

Results

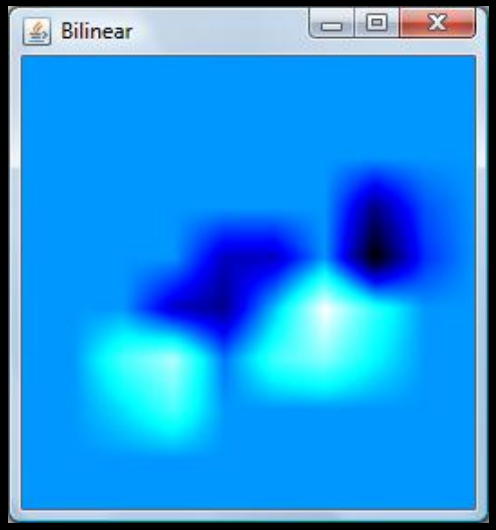


Function Approximator: Adaptation Demo

This shows each method after a single presentation of each of six patterns, three positive, three negative. What do you notice?



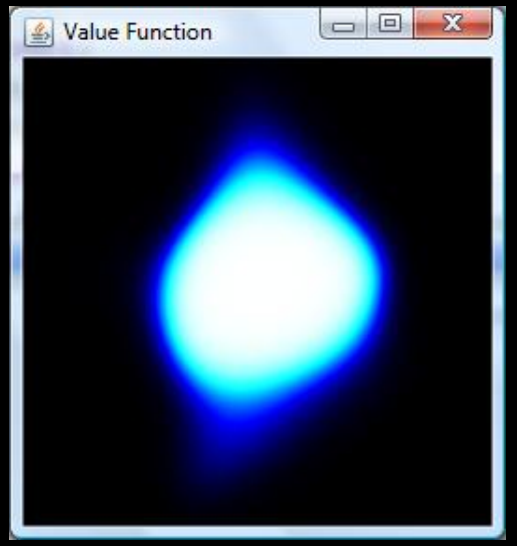
Play MLP Video

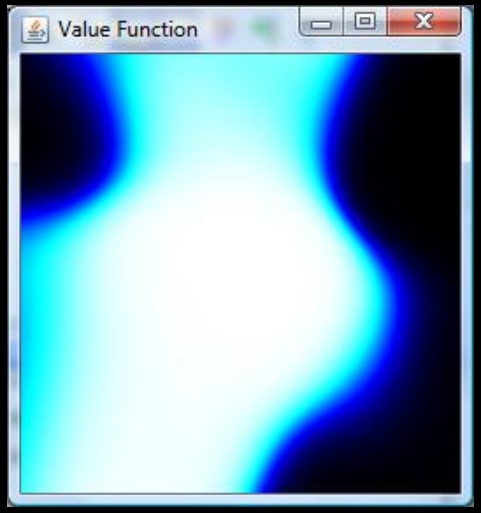


Play interpolated table video

Grid World – Evolved MLP

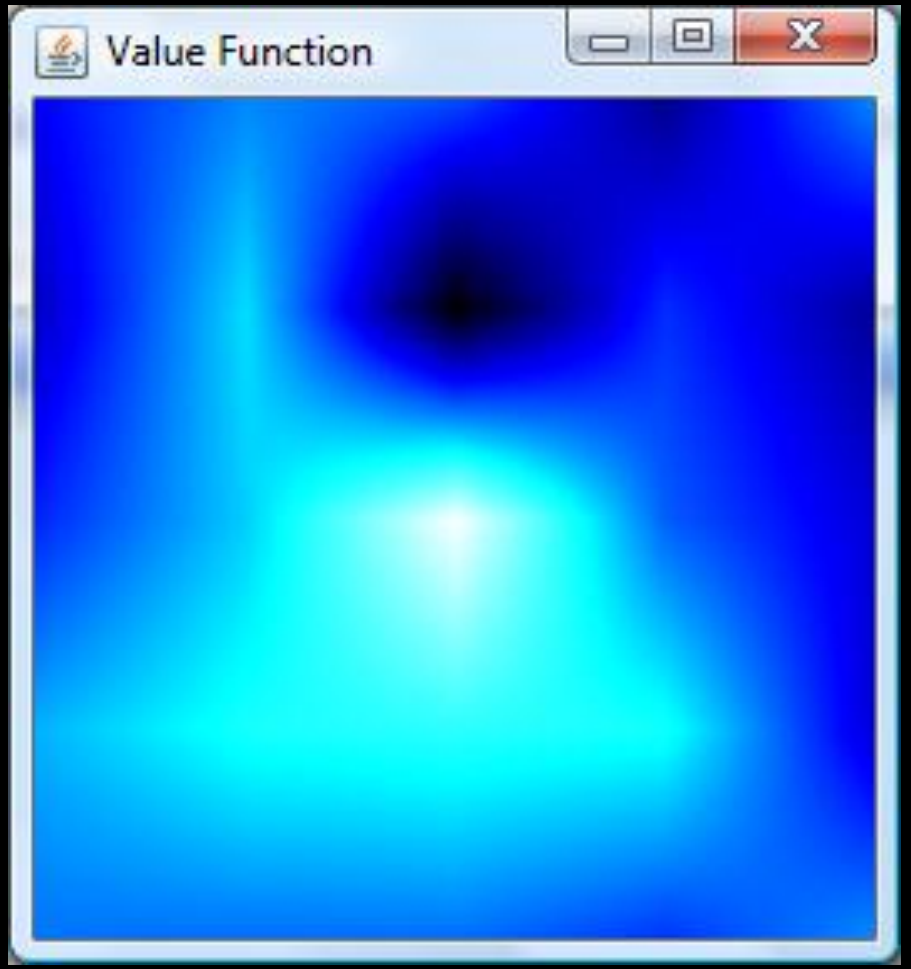
- MLP evolved using CMA-ES
- Gets close to optimal after a few thousand fitness evaluations
- Each one based on 25 episodes
- Needs tens of thousands of episodes to learn well
- Value functions may differ from run to run





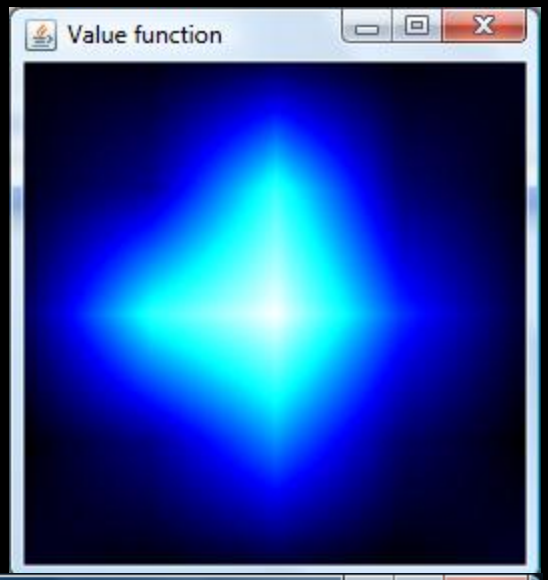
Evolved Interpolated Table

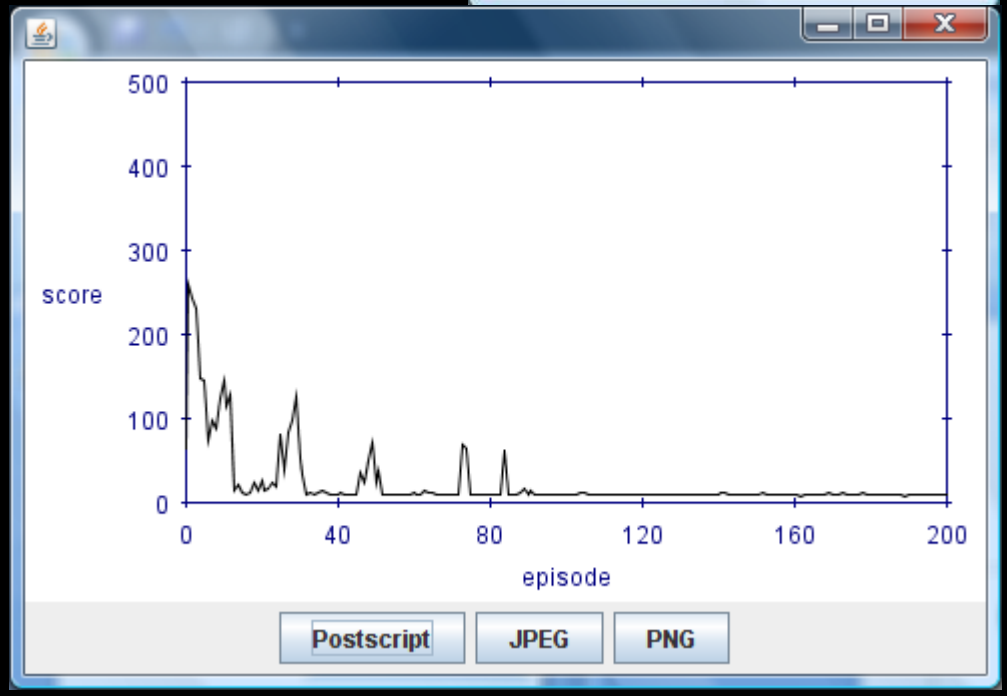
- A 5 x 5 interpolated table was evolved using CMA-ES, but only had a fitness of around 80
- Evolution does not work well with table functions in this case



TDL Again

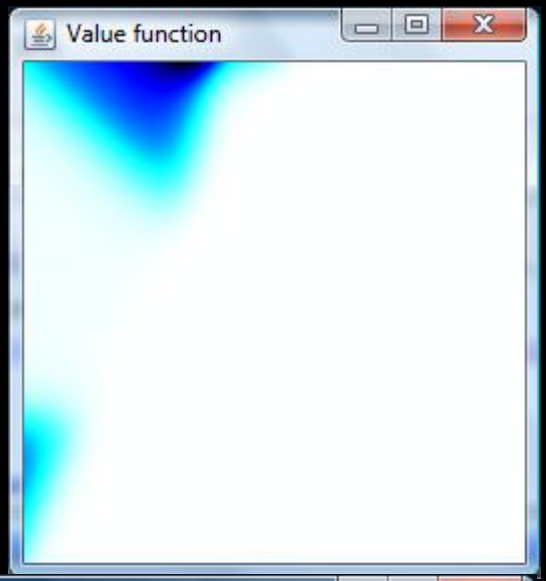
- Note how quickly it converges with the small grid
- Excellent performance within 100 episodes

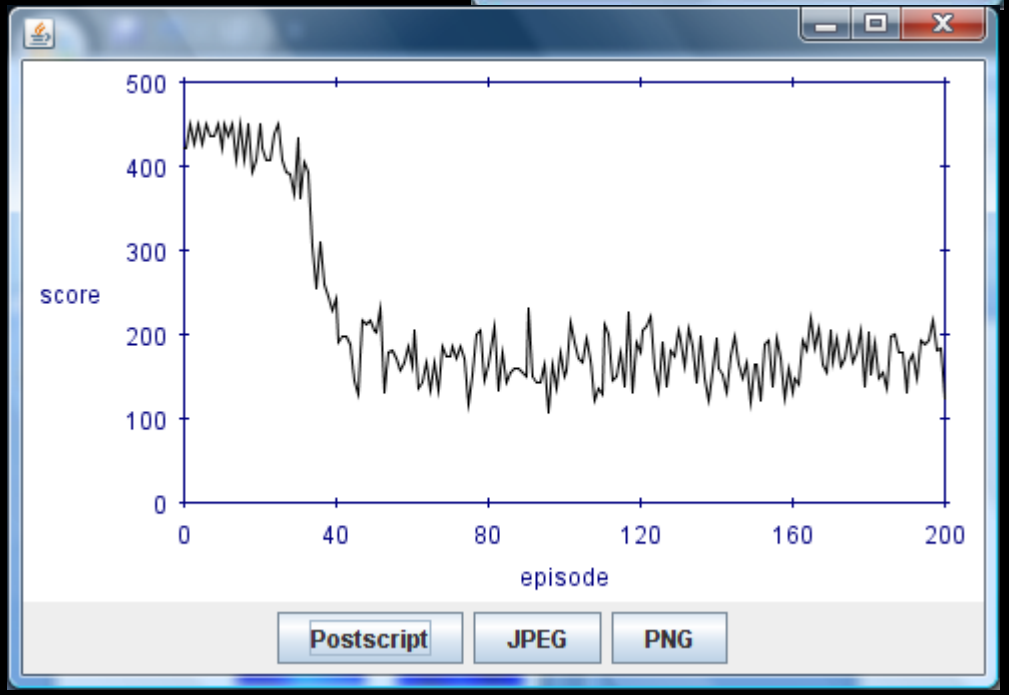




TDL MLP

- Surprisingly hard to make it work!





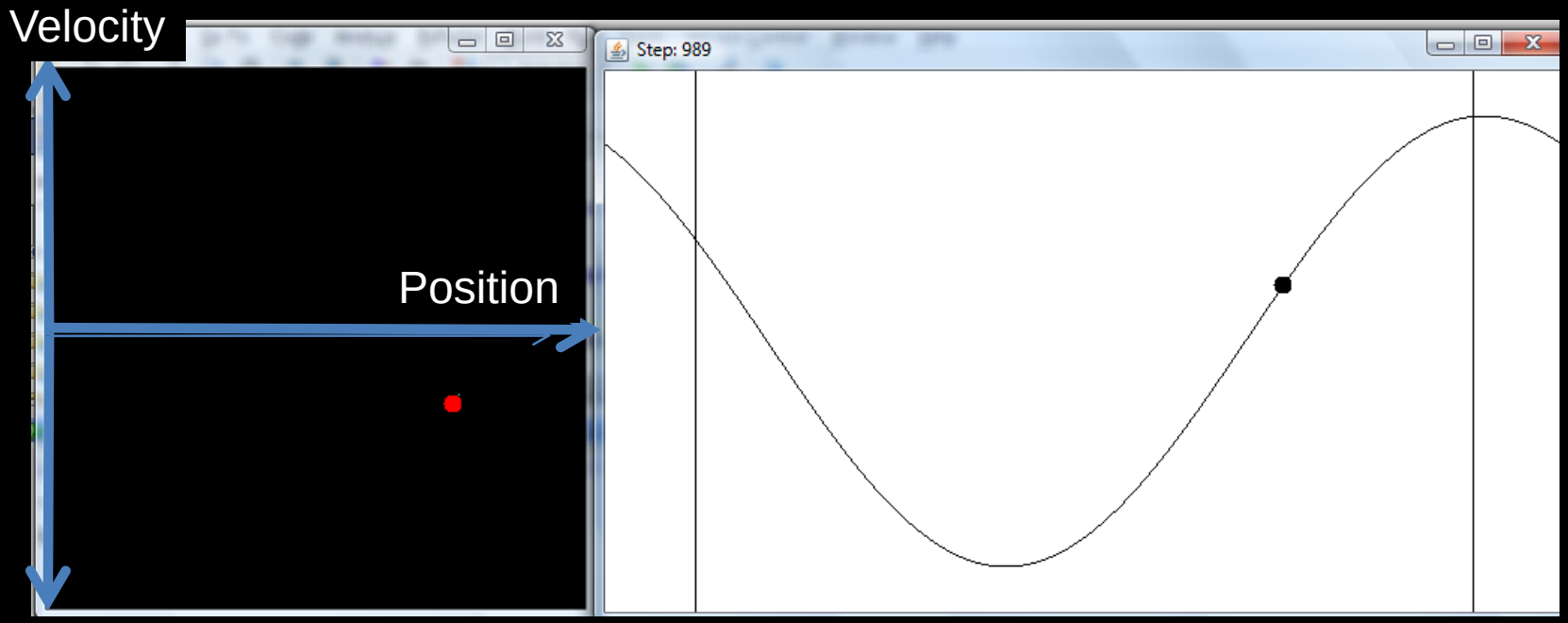
Grid World Results
Architecture x Learning Algorithm

- Interesting!
- The MLP / TDL combination is very poor
- Evolution with MLP gets close to TDL performance with N-Linear table, but at much greater computational cost

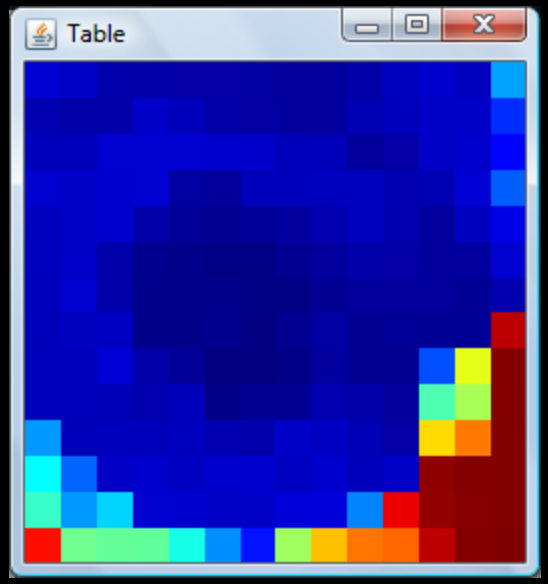
Architecture	Evolution (CMA-ES)	TDL(0)
MLP (15 hidden units)	**9.0**	126.0
Interpolated table (5 x 5)	11.0	**8.4**

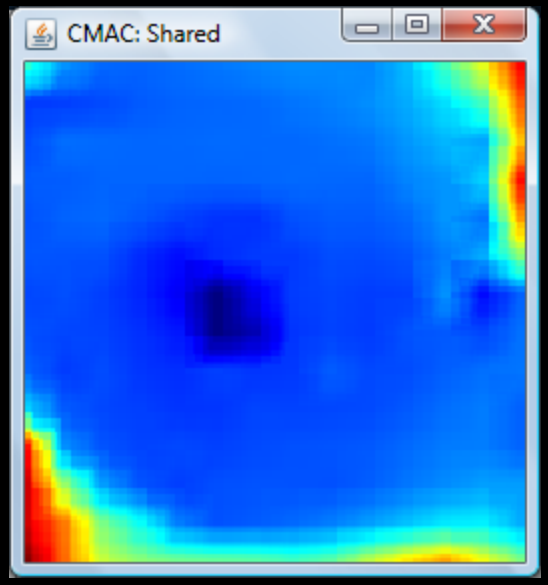
Simple Continuous Example: Mountain Car

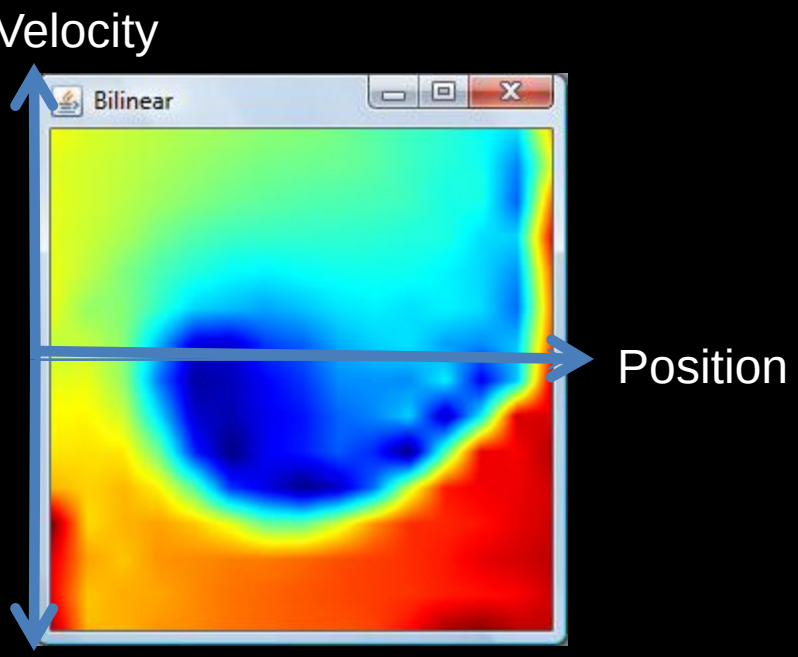
- Standard reinforcement learning benchmark
- Accelerate a car to reach goal at top of incline
- Engine force weaker than gravity



Value Functions Learned (TDL)







TDL Interpolated Table Video

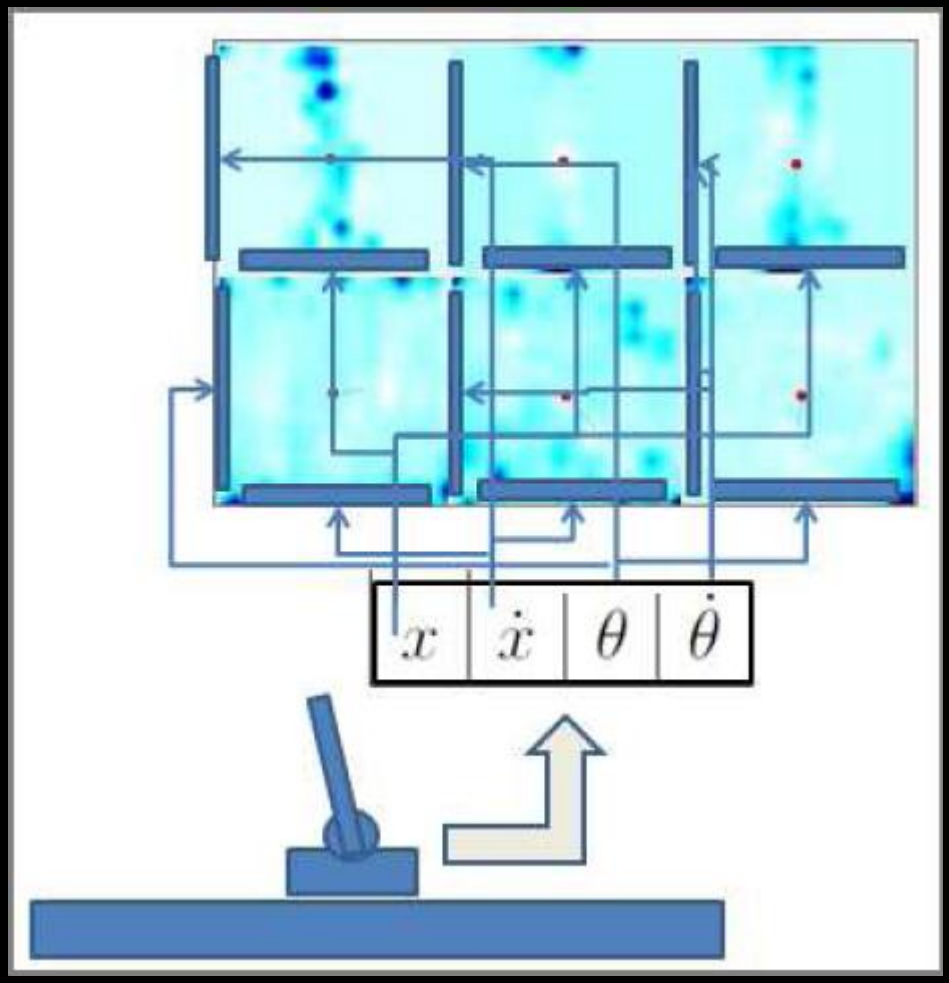
- Play video to see TDL in action, training a 5 x 5 table to learn the mountain car problem

Mountain Car Results (TDL, 2000 episodes, 15 x 15 tables, average of 10 runs)

System	Mean steps to goal (s.e.)
Naïve Table	1008 (143)
CMAC	60.0 (2.3)
Bilinear	**50.5 (2.5)**

Interpolated N-Tuple Networks (with Aisha A. Abdullahi)

- Use ensemble of N-linear look-up tables
  - Generalisation of bi-linear interpolation
- Sub-sample high dimensional input spaces
- Pole-balancing example:
  - 6 2-tuples



IN-Tuple Networks
Pole Balancing Results

TABLE 4. Number of attempts (evaluations) required to learn to balance a single pole given complete state information. All results apart from IN-Tuple taken from [38].

Method	Evaluations
IN-Tuple	44
CoSyNE	98
CMA-ES	283
SARSA-CMAC	540

Function Approximation Summary

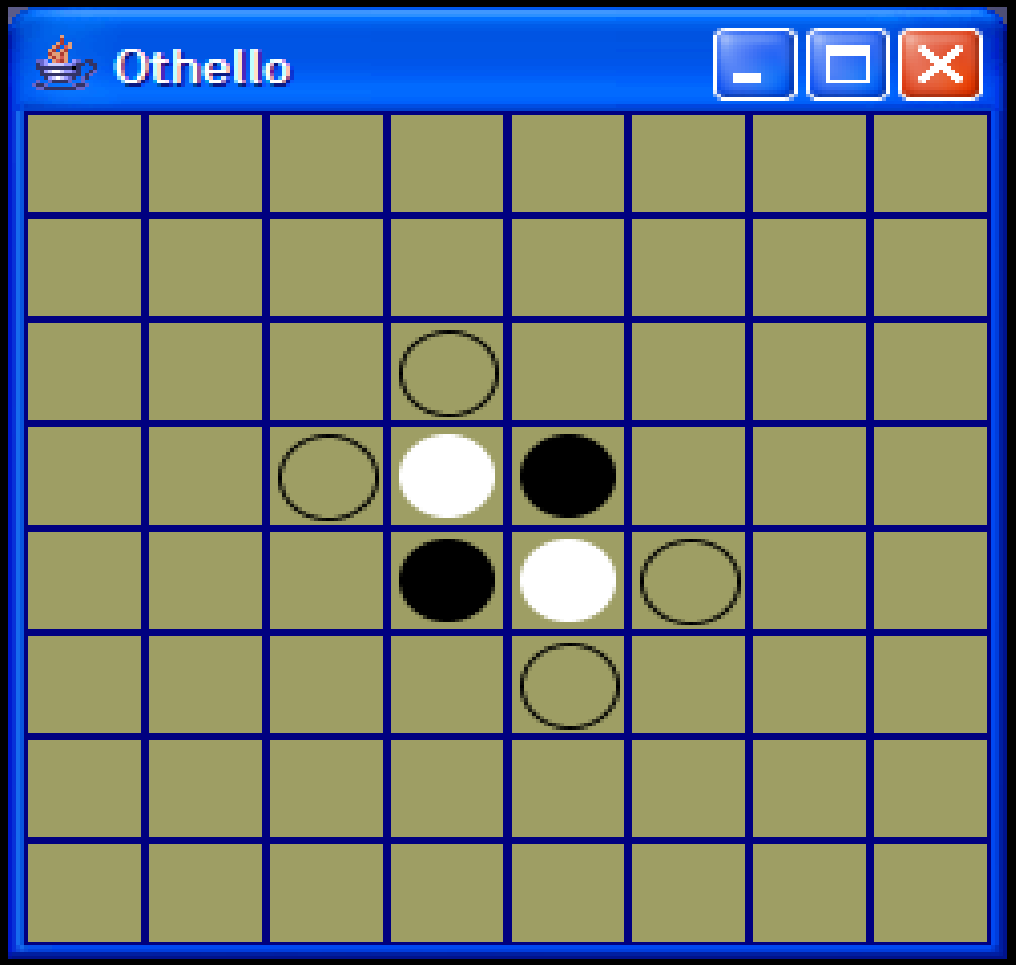
- The choice of function approximator has a critical impact on the performance that can be achieved
- It should be considered in conjunction with the learning algorithm
  - MLPs or global approximators work well with evolution
  - Table-based or local approximators work well with TDL
  - Further reading see: [InterpolatedTables]

Othello

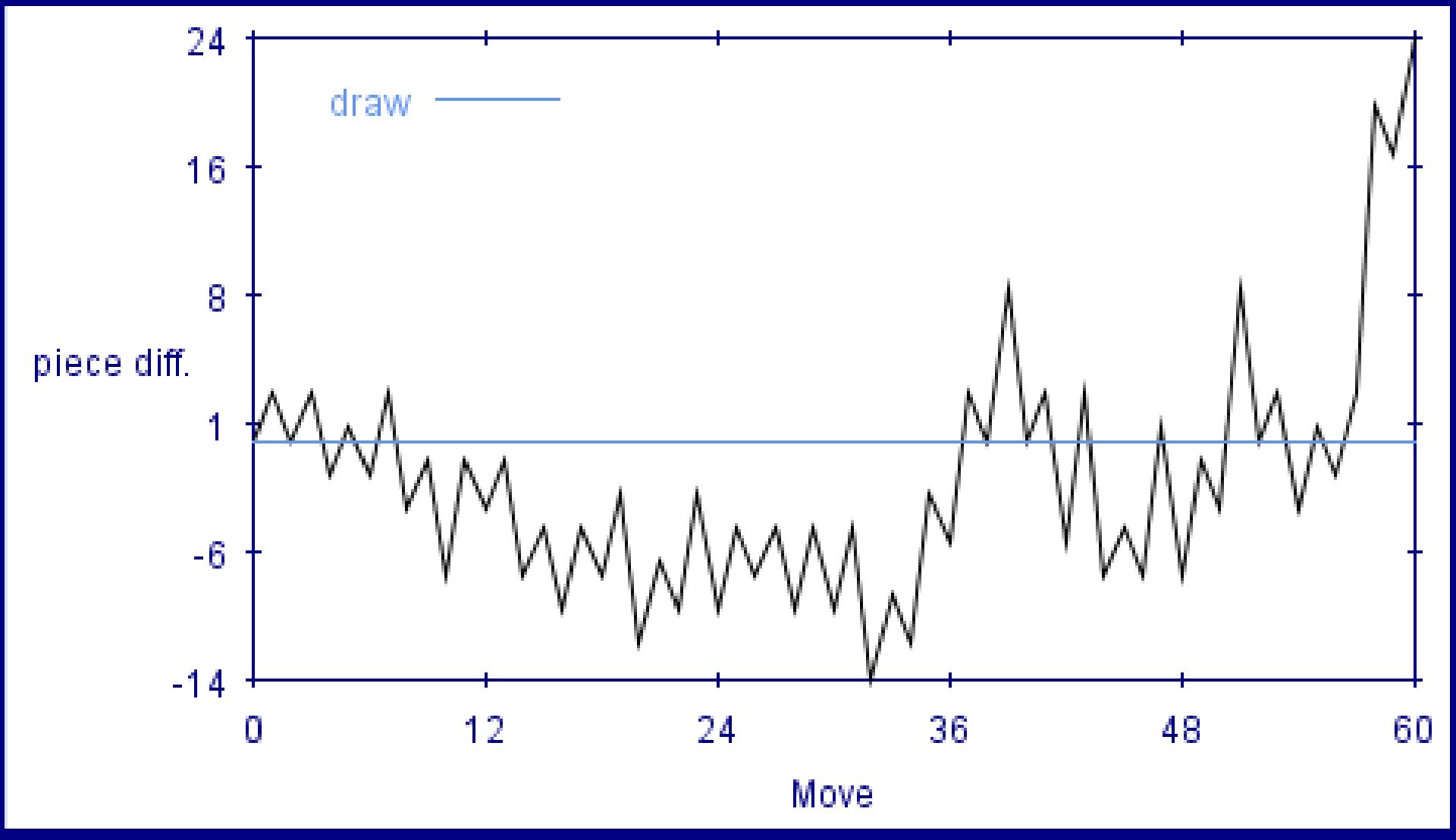
Othello

(from initial work done with Thomas Runarsson [CoevTDLOthello])

See Video



Volatile Piece Difference

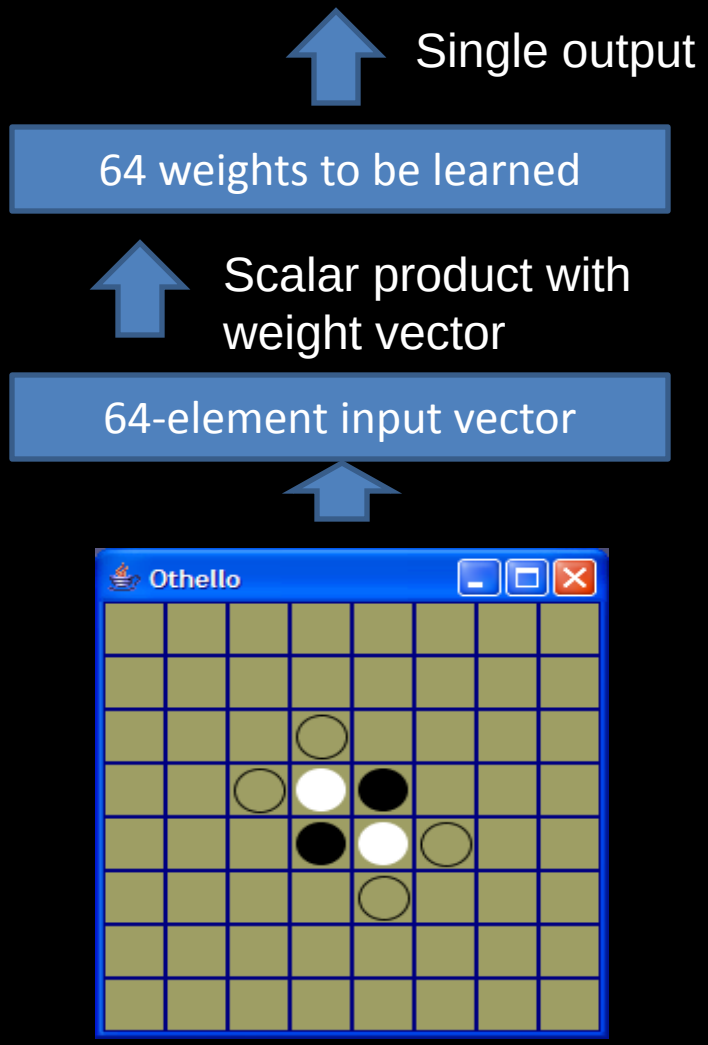


Learning a Weighted Piece Counter

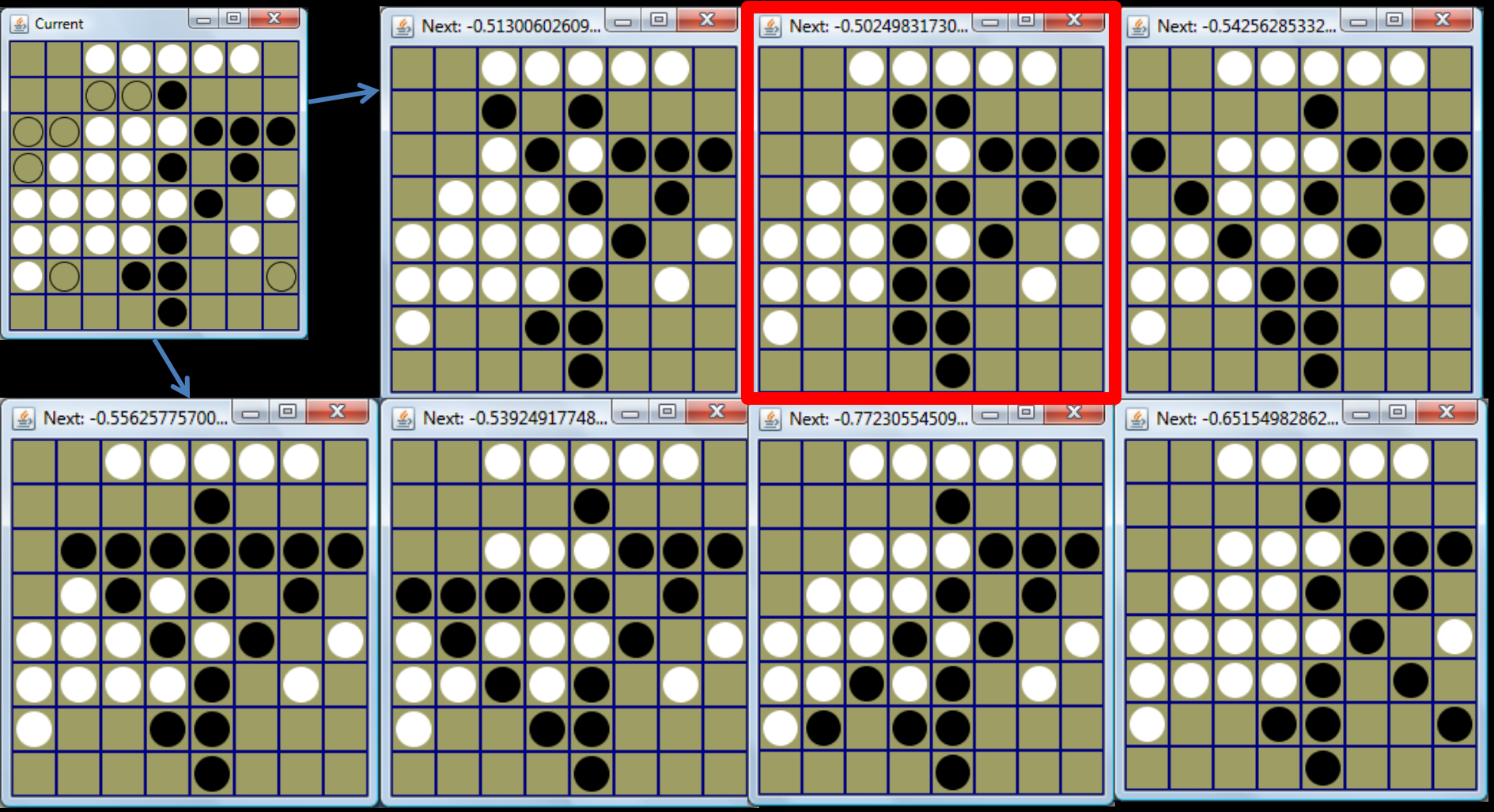
- Benefits of weighted piece counter
  - Fast to compute
  - Easy to visualise
  - See if we can beat the ‘standard’ weights
- Limit search depth to 1-ply
  - Enables many of games to be played
  - For a thorough comparison
  - Ply depth changes nature of learning problem
- Focus on machine learning rather than game-tree search
- Force random moves (with prob. 0.1)
  - Get a more robust evaluation of playing ability

Weighted Piece Counter

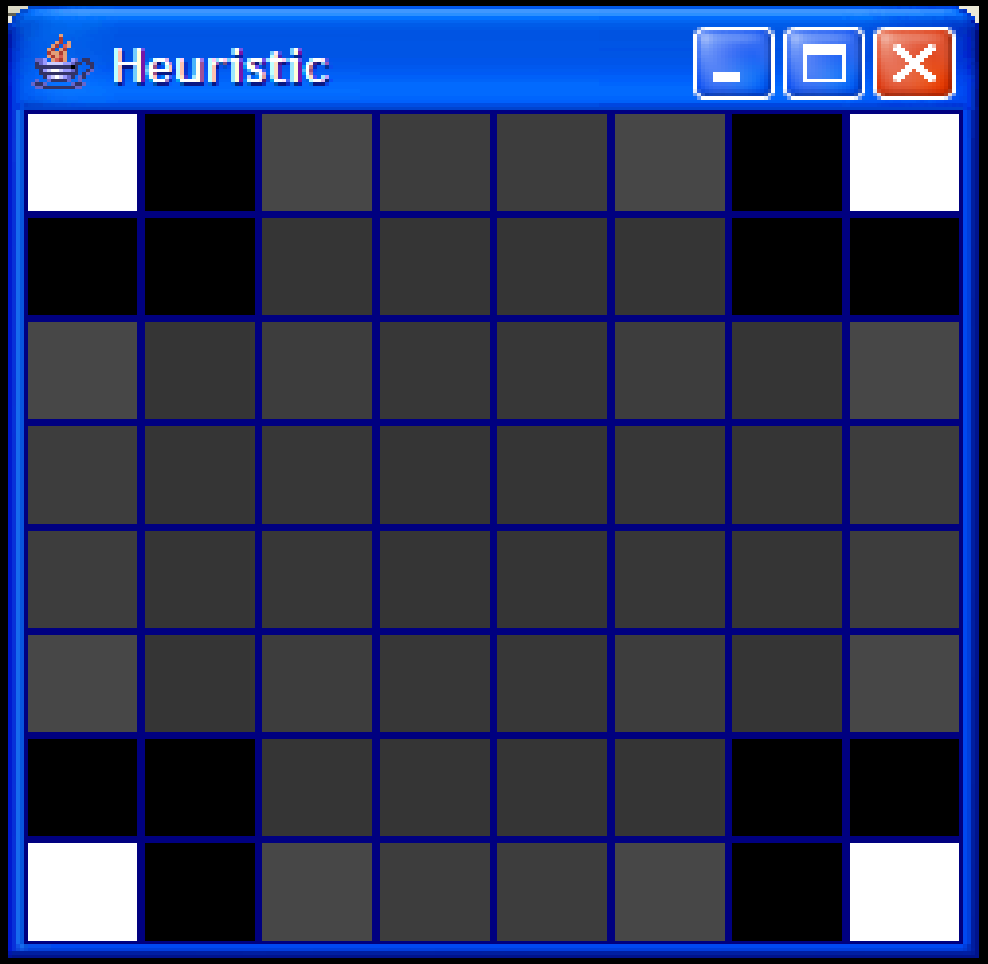
- Unwinds 8 x 8 board as a 64 dimensional input vector
- Each element of vector corresponds to a board square
- value of +1 (black), 0 (empty), -1 (white)



Othello: After-state Value Function



Standard “Heuristic” Weights (lighter = more advantageous)



TDL Algorithm

- Nearly as simple to apply as CEL

```
public interface TDLPlayer extends Player {
    void inGameUpdate(double[] prev, double[] next);
    void terminalUpdate(double[] prev, double tg);
}
```

$$\alpha\left[v(\boldsymbol{x}') - v(\boldsymbol{x})\right]\left(1 - v(\boldsymbol{x})^2\right)x_i$$

$$\alpha\left[r - v(\boldsymbol{x})\right]\left(1 - v(\boldsymbol{x})^2\right)x_i$$

- Reward signal only given at game end
- Initial alpha and alpha cooling rate tuned empirically

TDL in Java

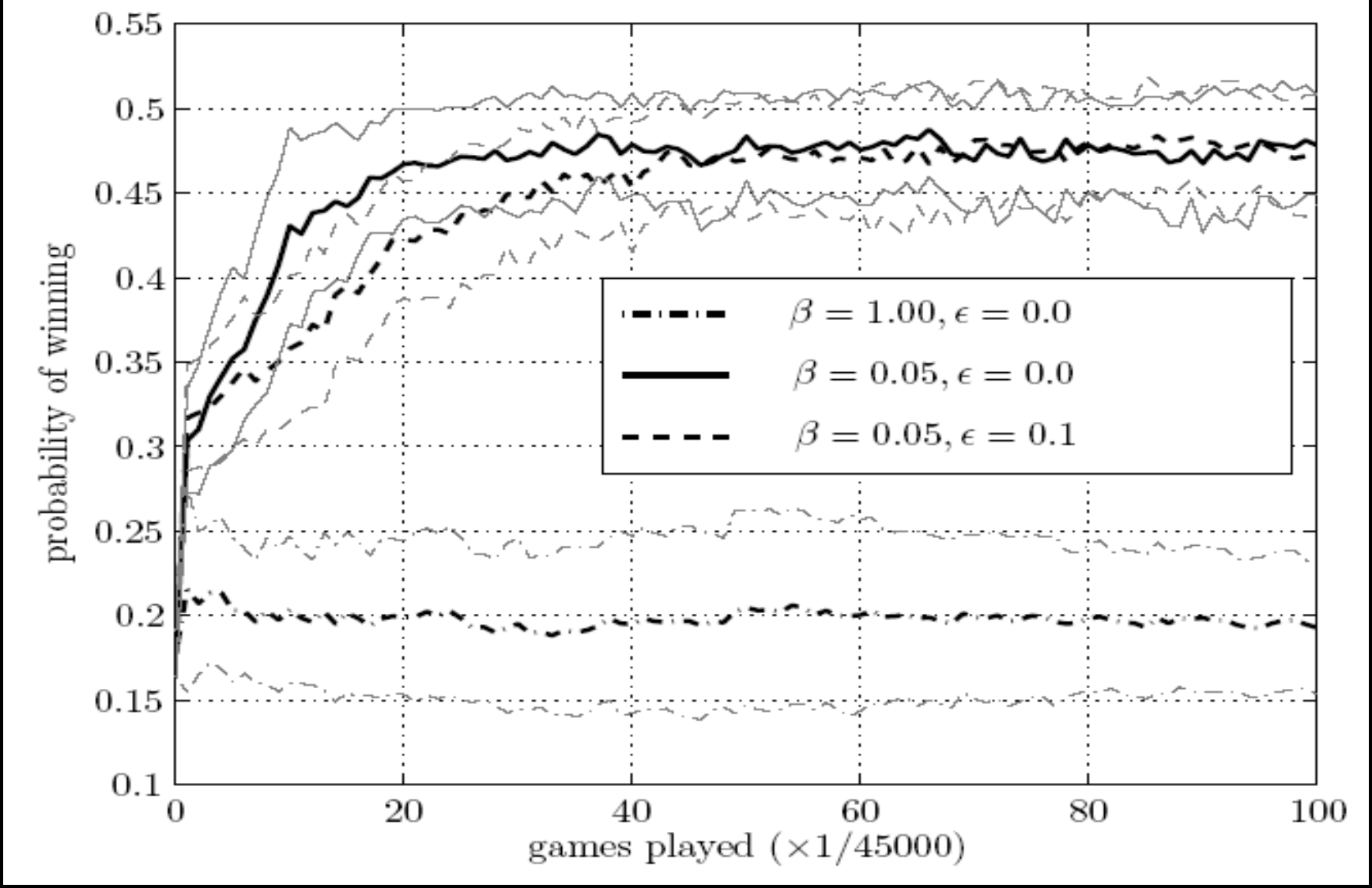
```
public void inGameUpdate(double[] prev, double[] next) {
    double op = tanh(net.forward(prev));
    double tg = tanh(net.forward(next));
    double delta = alpha * (tg - op) * (1 - op * op);
    net.updateWeights(prev, delta);
}

public void terminalUpdate(double[] prev, double tg) {
    double op = tanh(net.forward(prev));
    double delta = alpha * (tg - op) * (1 - op * op);
    net.updateWeights(prev, delta);
}
```

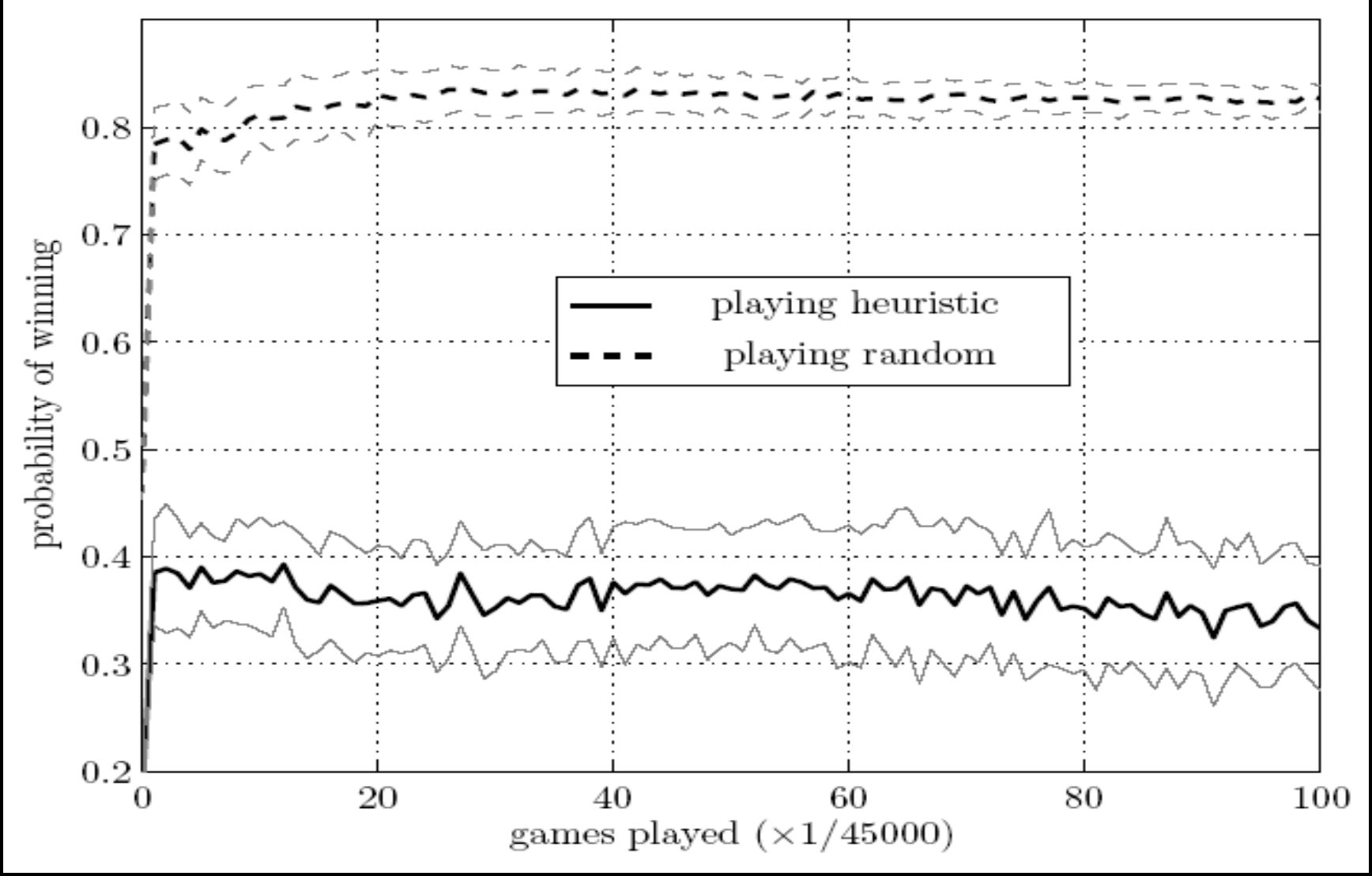
CEL Algorithm

- Evolution Strategy (ES)
  - (1, 10) (non-elitist worked best)
- Gaussian mutation
  - Fixed sigma (not adaptive)
  - Fixed works just as well here
- Fitness defined by full round-robin league performance (e.g. 1, 0, -1 for w/d/l)
- Parent child averaging
  - Defeats noise inherent in fitness evaluation
  - High Beta weights more toward best child
  - We found low beta works best – around 0.05

ES (1,10) v. Heuristic

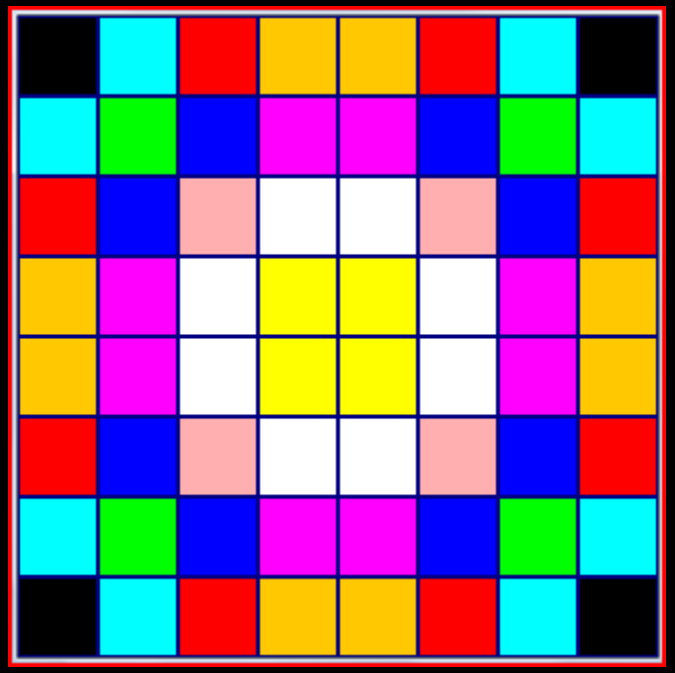


TDL v. Random and Heuristic



Better Learning Performance

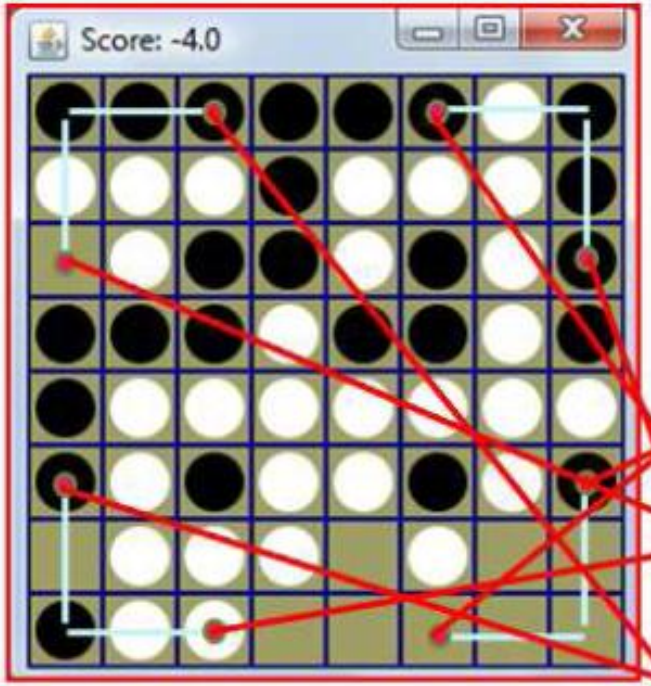
- Enforce symmetry
  - This speeds up learning



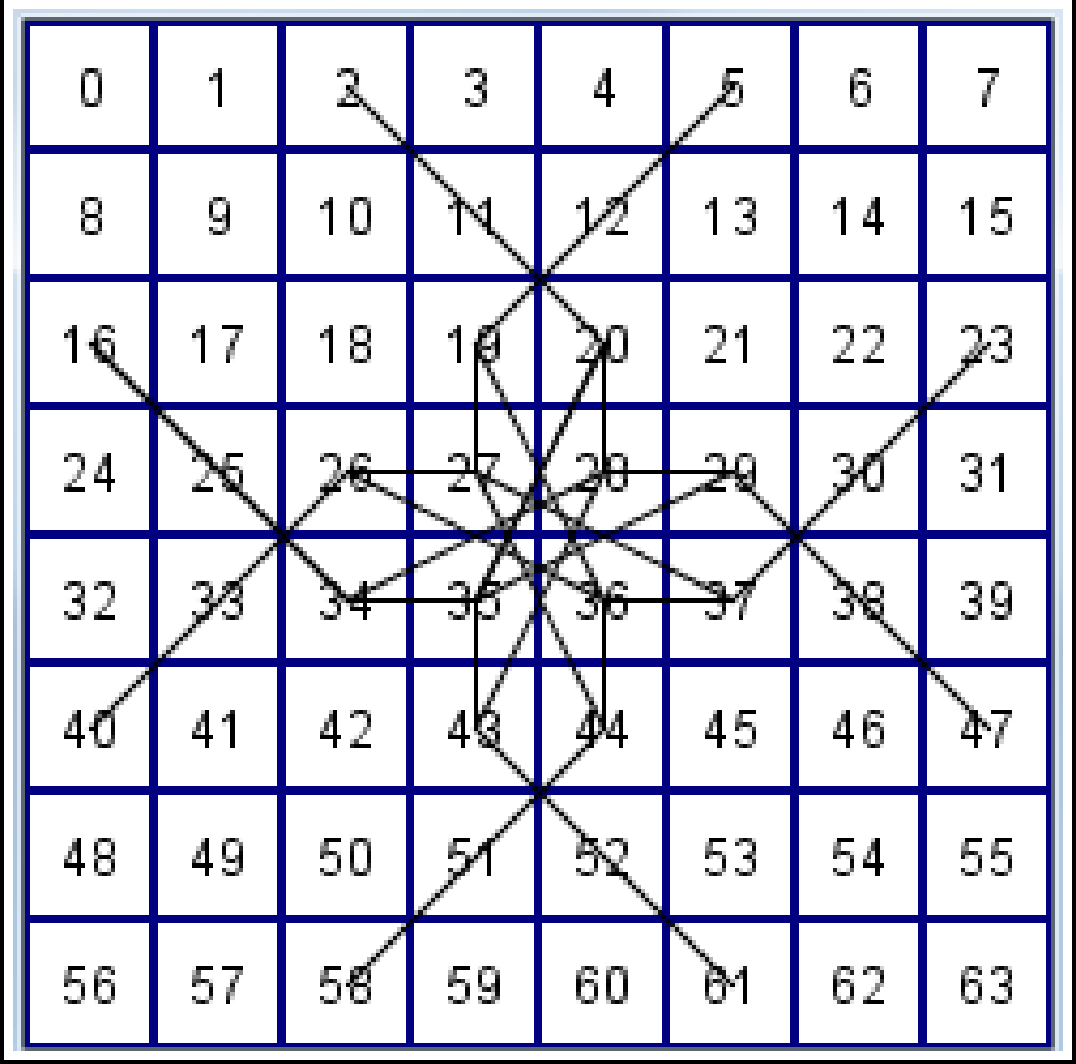
- Use **N-Tuple System** for value approximator [OthelloNTuple]

Symmetric 3-tuple Example

$$v(b) = \sum_{d \in D(b)} l(d)$$



Symmetric N-Tuple Sampling

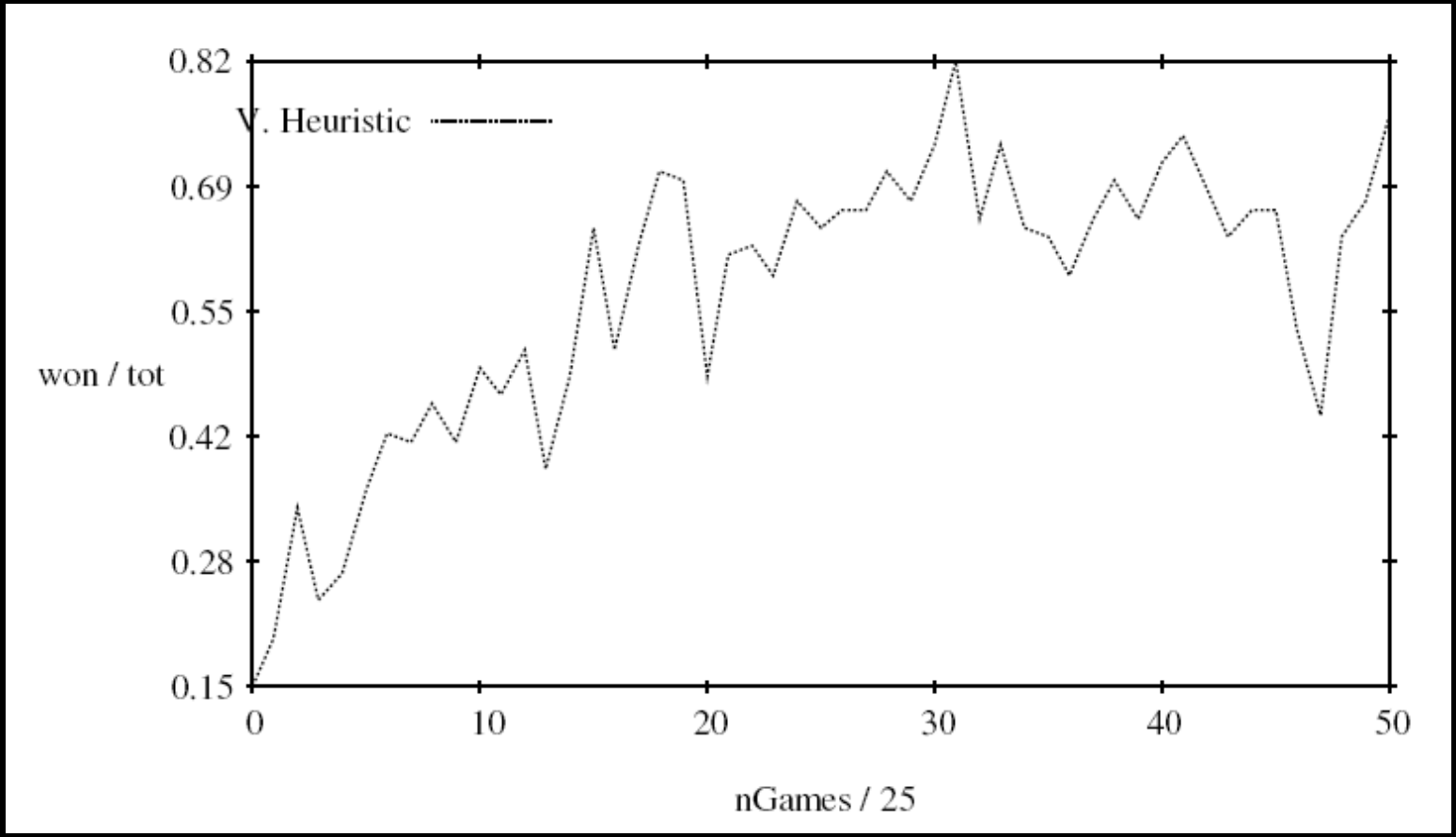


N-Tuple System

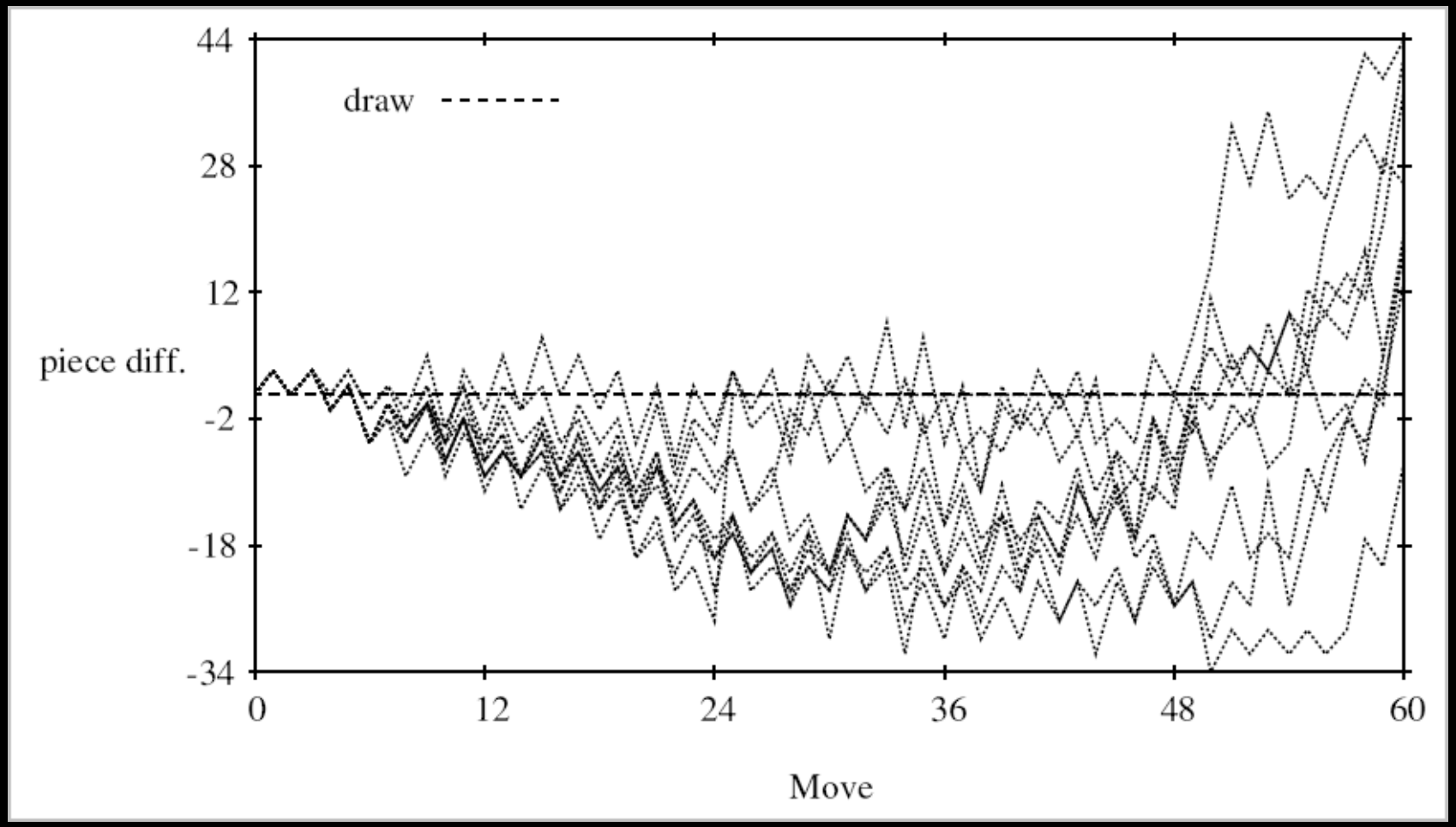
- Results used 30 random n-tuples
- Snakes created by a random 6-step walk
  - Duplicates squares deleted
- System typically has around 15000 weights
- Simple training rule:

$$l(d) = l(d) + \delta \quad \forall\, d \in D(b)$$

NTuple System (TDL)
total games = 1250
(very competitive performance)



Typical Learned strategy…
(N-Tuple player is +ve – 10 sample games shown)



Web-based League
(May 15th 2008)
All Leading entries are N-Tuple based

Trial League						
Position	**Name**	**Played**	**Won**	**Drawn**	**Lost**	**Format**
1	t15x6x8	100	79	3	18	SNT-Text
2	x30x6x8	100	71	4	25	SNT-Text
3	Stunner	100	67	1	32	SNT-Text
4	Woxy SNT	100	67	1	32	NET-WOX
5	WOX Test	100	65	1	34	NET-WOX
6	WOX Test 3	100	64	1	35	NET-WOX
7	newp8	100	64	3	33	SNT-Text
8	yp278a	100	64	2	34	SNT-Text
9	Stunner-2	100	63	6	31	SNT-Text
10	WOX Test 2	100	62	4	34	NET-WOX
11	MLP_Original-MoreNeurons.0.1-gen312-ties0.FF	100	60	4	36	MLP-Text
12	try3MLP_Original-MoreNeurons.0.1-gen341-ties0.FF	100	59	4	37	MLP-Text
13	shrd-MaxSolve-7c1kg	100	59	2	39	MLP-Text
14	test-mlp1	1000	582	34	384	unknown

Results versus IEEE CEC 2006 Champion (a manual EVO / TDL hybrid MLP)

n_{sp}	Won	Drawn	Lost
250	89	5	106
500	135	6	59
750	142	5	53
1000	136	2	62
1250	142	5	53

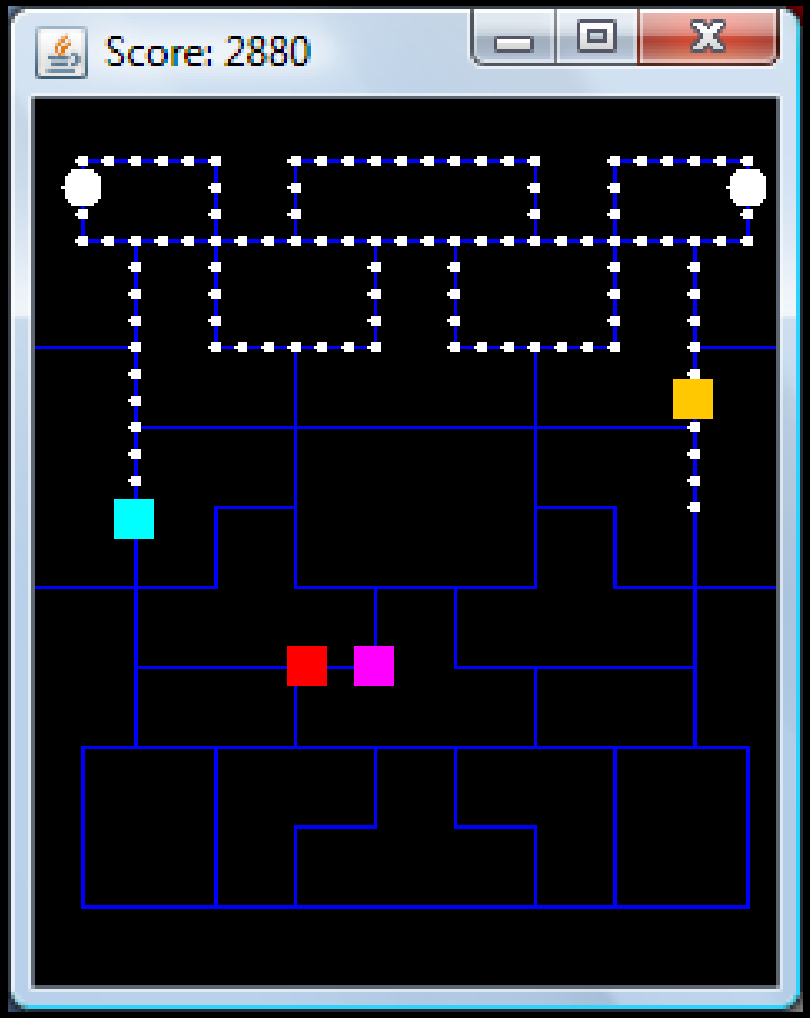
N-Tuple Summary

- Outstanding results compared to other game-learning architectures such as MLP
- May involve a very large number of parameters
- Temporal difference learning can learn these effectively
- But co-evolution fails (results not shown in this presentation)
  - Further reading: [OthelloNTuple]

Ms Pac-Man

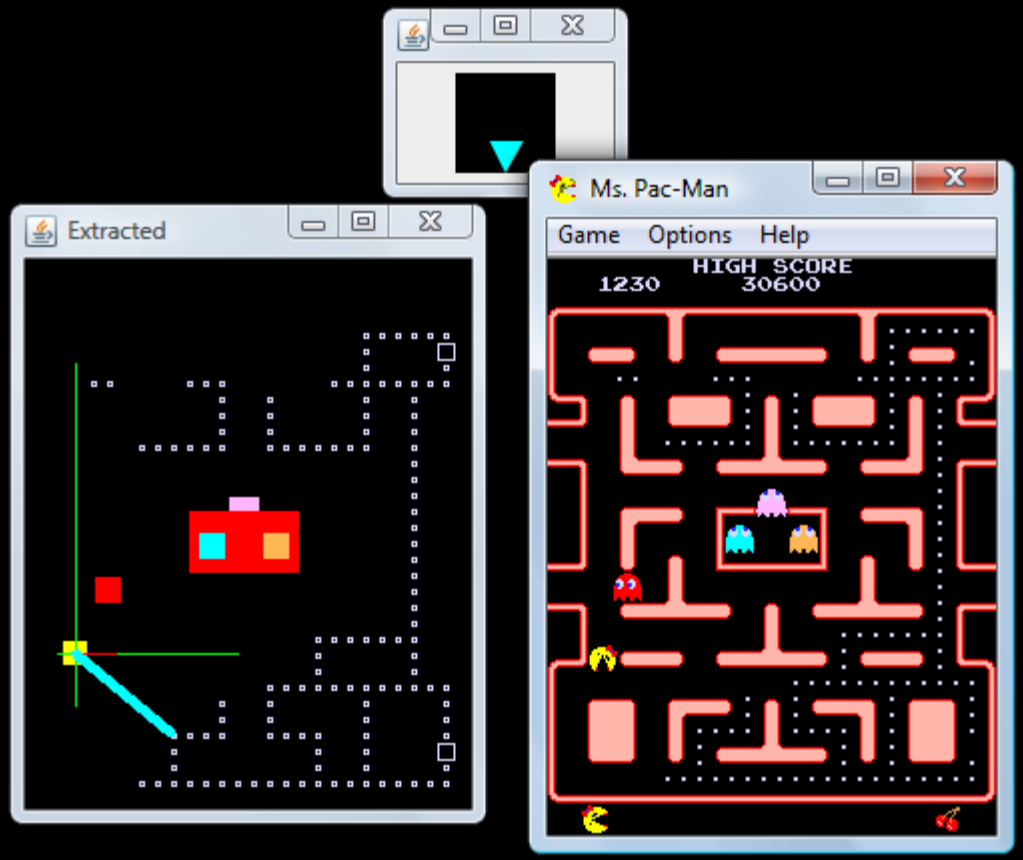
Ms Pac-Man

- Challenging Game
- Discrete but large state space
- Need to perform feature extraction to create input vector for function approximator



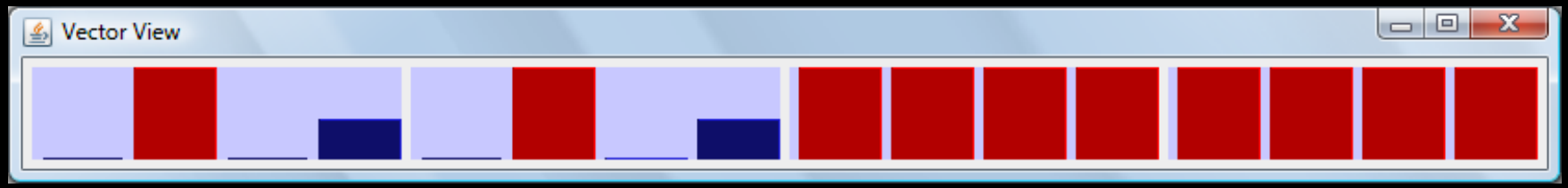
Screen Capture Mode

- Allows us to run software agents original game
- But simulated copy (previous slide) is much faster, and good for training
- Play Video of WCCI 2008 Champion
- The best computer players so far are largely hand-coded



Ms Pac-Man: Sample Features

- Choice of features are important
- Sample ones:
  - Distance to nearest ghost
  - Distance to nearest edible ghost
  - Distance to nearest food pill
  - Distance to nearest power pill
- These are displayed for each possible successor node from the

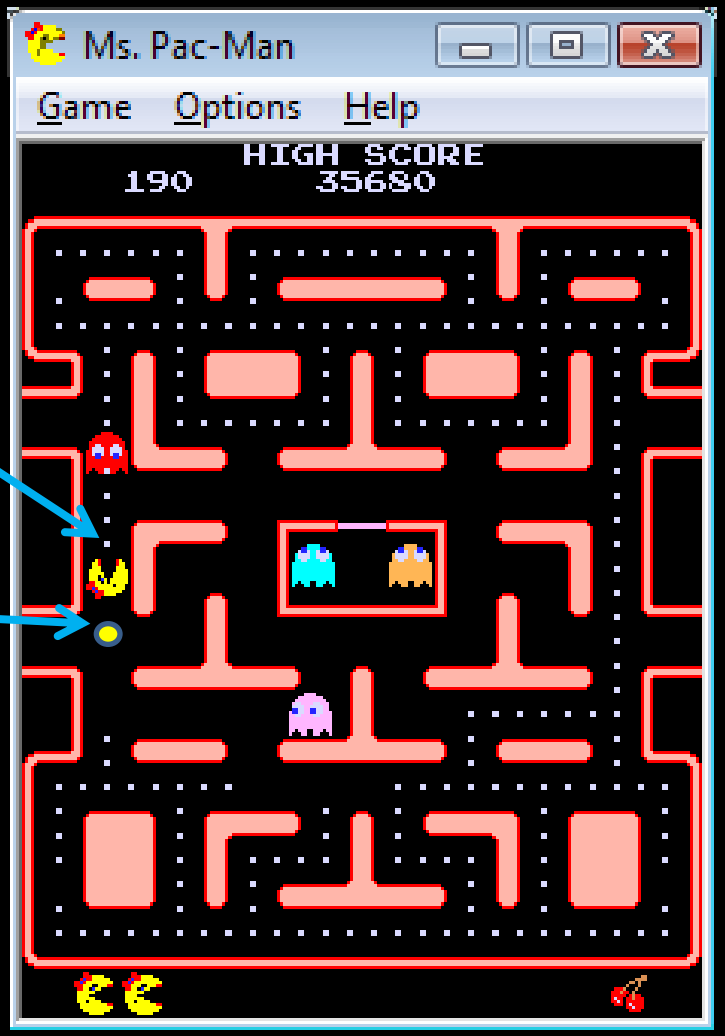


Results: MLP versus Interpolated Table

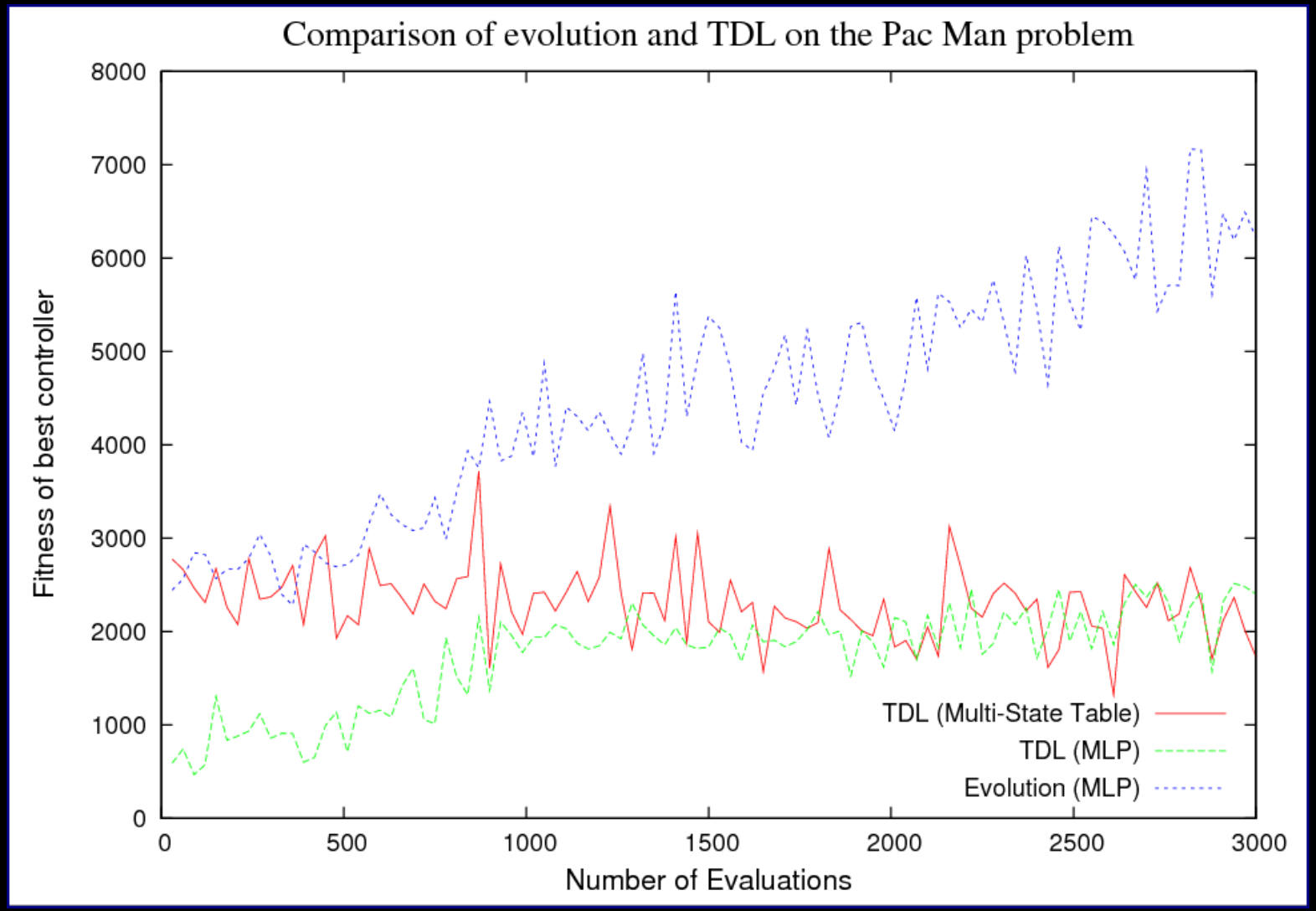
- Both used a 1+9 ES, run for 50 generations
- 10 games per fitness evaluation
- 10 complete runs of each architecture
- MLP had 5 hidden units
- Interpolated table had 3^4 entries
- So far each had a mean best score of approx 3,700
- Can we do better?

Alternative Pac-Man Features

- Uses a smaller feature space
- Distance to nearest pill
- Distance to nearest safe junction
- See: [BurrowPacMan]



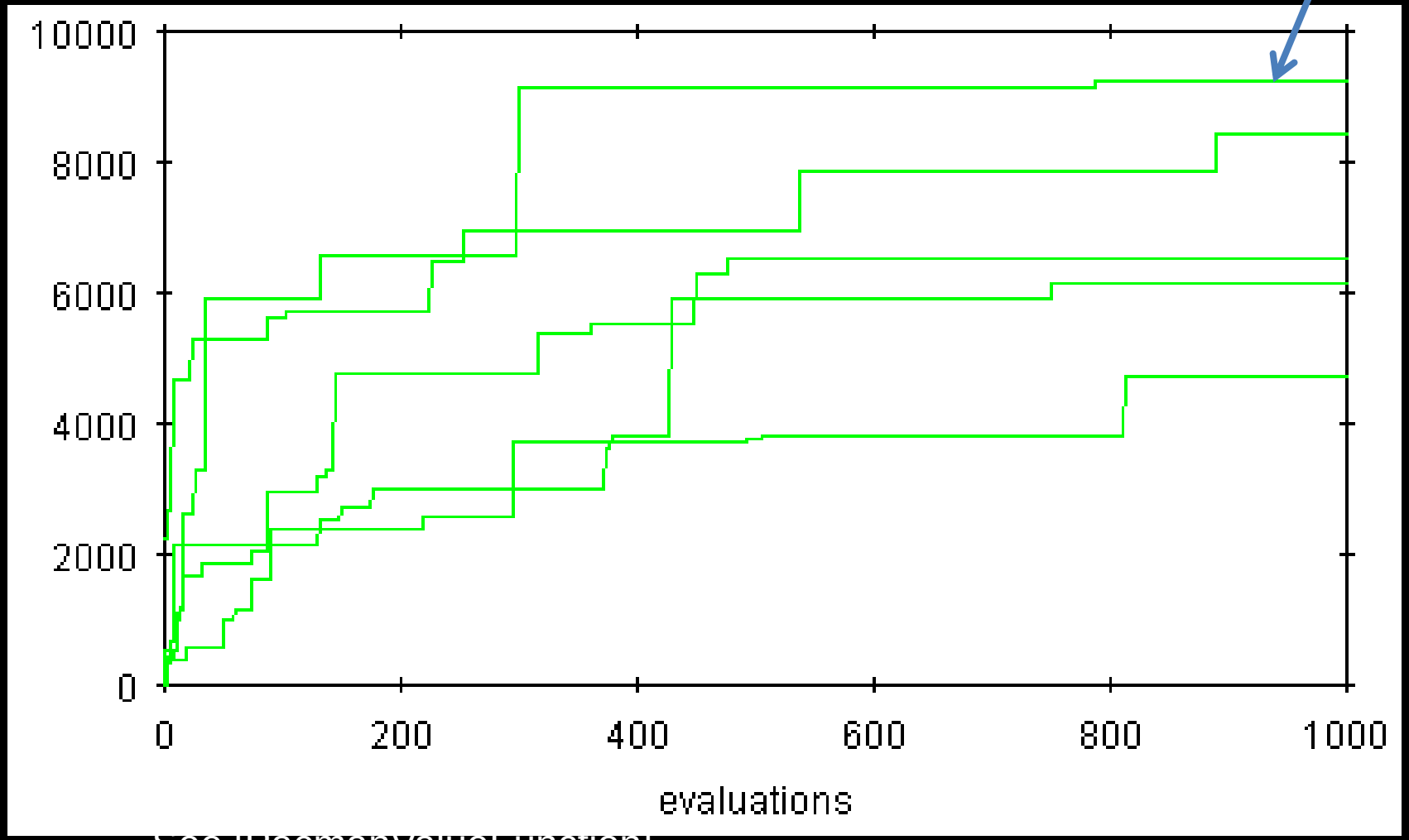
So far: Evolved MLP by far the best!



Importance of Noise / Non-determinism

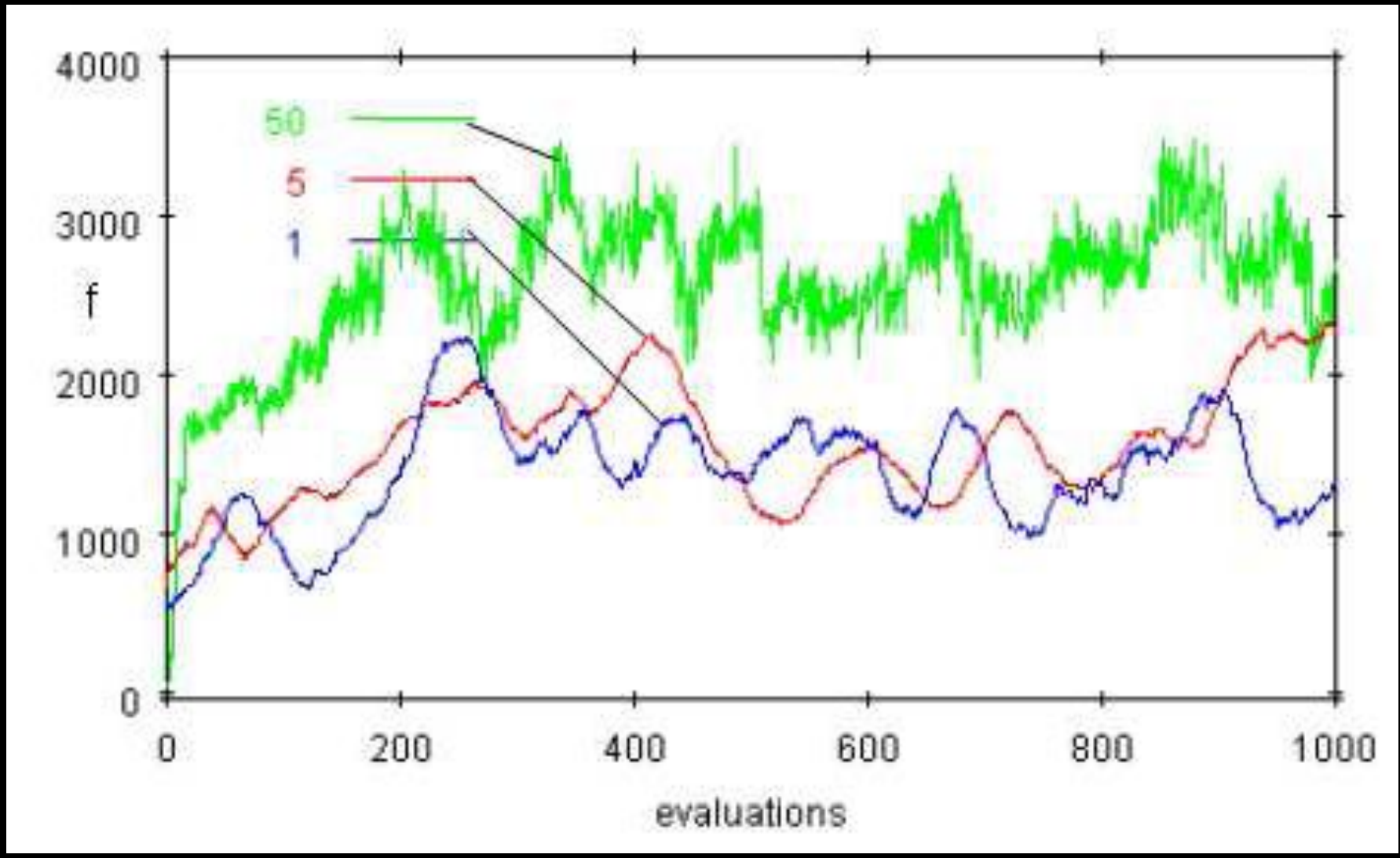
- When testing learning algorithms on games (especially single player games)
- Important that they are non-deterministic
- Otherwise evolution may evolve an implicit move sequence
- Rather than an intelligent behaviour
- Use an EA that is robust to noise
  - And always re-evaluate survivors

Evolved Perceptron: Deterministic Game



See [PacmanValueFunction]

Evolution on the Noisy Game



Pac-Man Summary

- Current computer players do not get close to expert human play
  - current computer champion ~25k
  - current human champion > 900k
- Also interesting from the point of view of the team of ghosts
- Alternative approaches:
  - rule-based (with learned priorities) [PacManRules]
  - tree-search [PacManTree]

Alternative Approaches

- This tutorial focussed on learning to play games where much of the emphasis is placed on the agent learning what to do given either the game state or a set of low-level features extracted from the game state
- And making actions at a direct level of play
- Also possible to give the agent a much higher-level set of inputs and outputs to work with
- This may achieve higher performance
  - at the expense of more human-led design
  - example: [PacManRules]

Monte Carlo Tree Search

- An alternative to learning is to do Monte-Carlo roll-outs from the current game state
- This enables an agent to be very far-sighted e.g. to play out until the end of the game until a result is certain
- Recent refinements of this approach have achieved phenomenal success in Go
- For more details see [MoGo]

General Game Playing

- When we test the ability of a machine learning algorithm to learn to play a game it's often unclear how much learning the algorithm has done, and how much expertise has been put in by the designer either intentionally or not
- General Game Playing is a fascinating idea: the agent is given the rules of the game in a type of first-order logic, and must then work out how to play it well [GGP]
- Interestingly, the current champion is based on Monte Carlo UCT [CadiaPlayer]

Summary

- All choices need careful investigation
  - Big impact on performance
- Function approximator
  - N-Tuples and interpolated tables: very promising
  - Table-based typically learns better than MLPs with TDL
- Learning algorithm
  - TDL is often better for large numbers of parameters
  - But TDL may perform poorly with MLPs
  - Evolution is easier to apply
- Learning to play games is hard – much more research needed
- I hope this tutorial has given you a good introduction to the main concepts

References-I

- **[TDGammon]** G. Tesauro, **Temporal difference learning and TD-gammon**, Communications of the ACM, vol. 38, no. 3, pp. 58 – 68, 1995.
- **[Neuro-Gammon]** J. Pollack and A. Blair, **Co-evolution in the successful learning of backgammon strategy**, Machine Learning, vol. 32, pp. 225 – 240, 1998.
- **[Blondie]** K. Chellapilla and D. Fogel, **Evolving neural networks to play checkers without expert knowledge**, IEEE Transactions on Neural Networks, vol. 10, no. 6, pp. 1382 – 1391, 1999.

References-II

- **[Menace]** D. Michie, **Trial and error**, In Science Survey, part 2, Penguin, 1961, pp. 129 – 145.
- **[NERO]** Kenneth O. Stanley, Bobby D. Bryant, Risto Miikkulainen, **Real-Time Evolution in the NERO Video Game**, Proceedings of IEEE CIG 2005
- **[RL]** R. Sutton and A. Barto, **Introduction to Reinforcement Learning**, MIT Press, 1998.

References-III

- [CMA-ES] Nikolaus Hansen, **The CMA Evolution Strategy: A Tutorial**, April 26 2008 URL : http://www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf
- [InfoRates] Simon M. Lucas, **Investigating Learning Rates for Evolution and Temporal Difference Learning**, IEEE Computational Intelligence and Games (2008)
- [InterpolatedTables] Simon M. Lucas, **Temporal Difference Learning with Interpolated Table Value Functions**, IEEE Computational Intelligence and Games (2009)
- [CoevTDLOthello] Simon M. Lucas and Thomas P. Runarsson, **Temporal Difference Learning Versus Co-Evolution for Acquiring Othello Position Evaluation**, IEEE Symposium on Computational Intelligence and Games (2006)
- [OthelloNTuple] Simon M. Lucas, **Learning to Play Othello with N-Tuple Systems**, Australian Journal of Intelligent Information Processing (2008), v. 4, pages: 1 - 20

References-IV

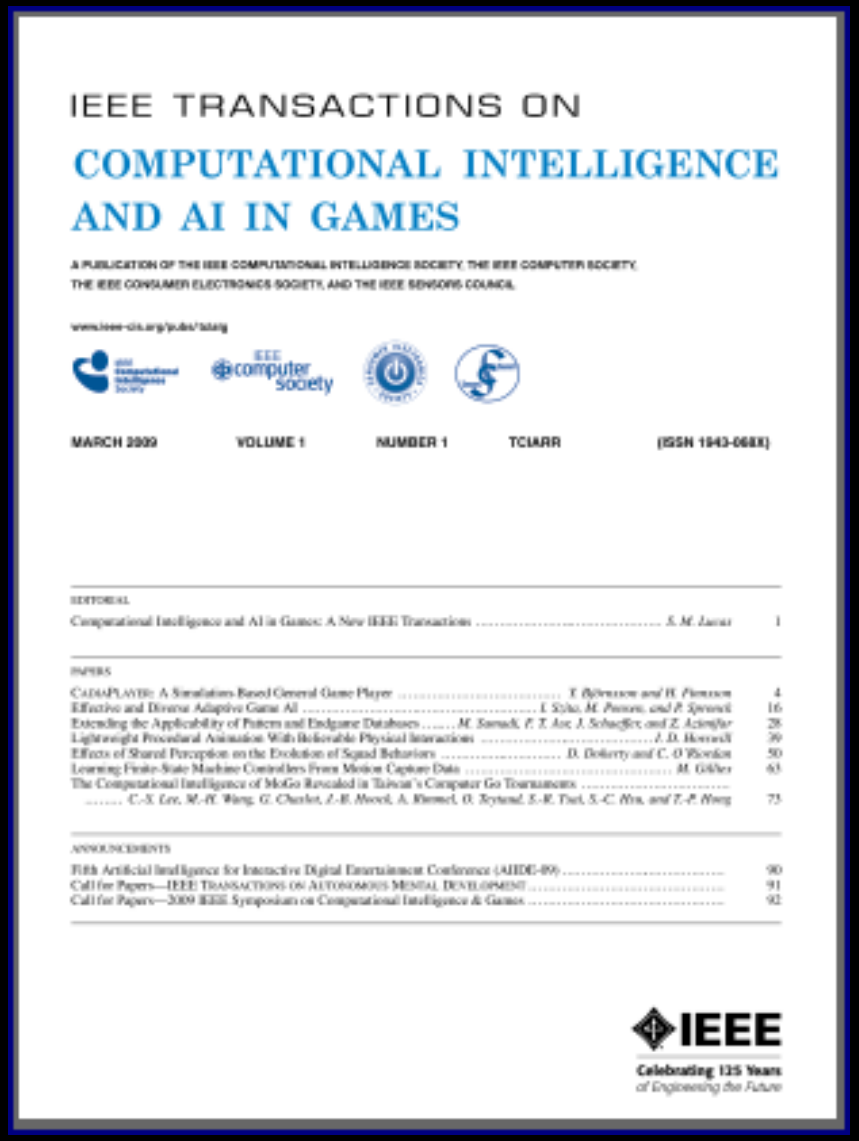
- [BurrowPacMan] Peter Burrow and Simon M. Lucas, **Evolution versus Temporal Difference Learning for learning to play Ms. Pac-Man**, IEEE Computational Intelligence and Games (2009)
- [PacManValueFunction] Simon M. Lucas, **Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man**, IEEE Symposium on Computational Intelligence and Games (2005)
- [PacManRules] I. Szita and A. Lorincz, **Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations through Ms. Pac-Man** Journal of AI Research, Volume 30, 2007, 659 - 684
- [PacManTree] David Robles and Simon M. Lucas, **A Simple Tree Search Method for Playing Ms. Pac-Man,** IEEE Computational Intelligence and Games (2009)

References-V

- [MoGo] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, **Modification of UCT with patterns in Monte-Carlo Go**, INRIA, Tech. Rep. Technical Report 6062, 2006.
- [GGP] M.R. Genesereth, N. Love, and B. Pell, **General game playing: Overview of the AAAI competition**, AI Magazine, no. 2, p. 62–72, 2005.
- [CadiaPlayer] Y. Bjornsson and H. Finnsson **CadiaPlayer: A Simulation-Based General Game** Player, IEEE Transactions on Computational Intelligence and AI in Games, Vol 1, 2009, p. 4-15

Places to publish (and to read)

- IEEE Transactions on Computational Intelligence and AI in Games
- Good conferences
  - IEEE CIG
  - AIIDE
- Special Sessions
  - At IEEE CEC, IEEE WCCI, and many other conferences



IEEE TRANSACTIONS ON

COMPUTATIONAL INTELLIGENCE AND AI IN GAMES

A PUBLICATION OF THE IEEE COMPUTATIONAL INTELLIGENCE SOCIETY, THE IEEE COMPUTER SOCIETY, THE IEEE CONSUMER ELECTRONICS SOCIETY, AND THE IEEE SENSORS COUNCIL

www.ieee-cis.org/pubs/tciaig

MARCH 2009 VOLUME 1 NUMBER 1 TCIARR (ISSN 1943-068X)

EDITORIAL

Computational Intelligence and AI in Games: A New IEEE Transactions ..... *S. M. Lucas* 1

PAPERS

CADIAPLAYER: A Simulation-Based General Game Player ..... *Y. Björnsson and H. Finnsson* 4
Effective and Diverse Adaptive Game AI ..... *I. Szita, M. Ponsen, and P. Spronck* 16
Extending the Applicability of Pattern and Endgame Databases ..... *M. Sumedi, P. T. Aue, J. Schaeffer, and Z. Aciemfer* 28
Lightweight Procedural Animation With Believable Physical Interactions ..... *J. D. Horswill* 39
Effects of Shared Perception on the Evolution of Squad Behaviors ..... *D. Doherty and C. O'Riordan* 50
Learning Finite-State Machine Controllers From Motion Capture Data ..... *M. Gildea* 63
The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments ..... *C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoorel, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong* 73

ANNOUNCEMENTS

Fifth Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-09) 90
Call for Papers—IEEE TRANSACTIONS ON AUTONOMOUS MENTAL DEVELOPMENT 91
Call for Papers—2009 IEEE Symposium on Computational Intelligence & Games 92

IEEE
Celebrating 125 Years
of Engineering the Future

Appendix:
(1,lambda) Evolution Strategy (ES)

```
INITIALIZE (Pop, λ individuals)
while running do
    for i=1 to (λ) do
        EVALUATE (Pop[i])
    end
    Sort on best fitness first, breaking ties randomly
    Parents are Pop[1] ... Pop[μ]
    for i=1 to (μ + λ) do
        Pop[i] ← MUTATED COPY (RANDOMLY SELECTED PARENT)
    end
end
```